IMA Journal of Applied Mathematics (2024) **89**, 143–174
https://doi.org/10.1093/imamat/hxae011
Advance Access publication on 23 May 2024

Can physics-informed neural networks beat the finite element method?

Tamara G. Grossmann[*]
Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK
[*]Corresponding author: t.g.grossmann@gmail.com

Urszula Julia Komorowska
Department of Computer Science and Technology, University of Cambridge, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK

Jonas Latz
Department of Mathematics, University of Manchester, Alan Turing Building, Oxford Road, Manchester M13 9PL, UK

and

Carola-Bibiane Schönlieb
Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK

[Received on 31 January 2023; revised on 12 April 2024; accepted on 16 May 2024]

Partial differential equations (PDEs) play a fundamental role in the mathematical modelling of many processes and systems in physical, biological and other sciences. To simulate such processes and systems, the solutions of PDEs often need to be approximated numerically. The finite element method, for instance, is a usual standard methodology to do so. The recent success of deep neural networks at various approximation tasks has motivated their use in the numerical solution of PDEs. These so-called physics-informed neural networks and their variants have shown to be able to successfully approximate a large range of PDEs. So far, physics-informed neural networks and the finite element method have mainly been studied in isolation of each other. In this work, we compare the methodologies in a systematic computational study. Indeed, we employ both methods to numerically solve various linear and nonlinear PDEs: Poisson in 1D, 2D and 3D, Allen–Cahn in 1D, semilinear Schrödinger in 1D and 2D. We then compare computational costs and approximation accuracies. In terms of solution time and accuracy, physics-informed neural networks have not been able to outperform the finite element method in our study. In some experiments, they were faster at evaluating the solved PDE.

Keywords: partial differential equations; finite element method; deep learning; physics-informed neural networks.

1. Introduction

Partial differential equations (PDEs) are a corner stone of mathematical modelling and possibly applied mathematics itself. They are used to model a multitude of physical (Lin & Segel, 1988), biological (Clifford Henry Taubes, 2008), socioeconomic (Burger *et al.*, 2014) and financial (Heston, 2015) systems and processes. Beyond classical modelling, PDEs can describe the evolution of certain stochastic processes (Risken, 1996), be used in image reconstruction (Rudin *et al.*, 1992), as well as for filtering (Kushner, 1964) and optimal control (Bellmann, 1954) of dynamical systems.

The underlying idea is always the same: we aim to represent some process through a function that describes the behaviour of the process in space and time. The PDE is then a collection of laws that this function is supposed to satisfy. An obvious questions is whether these laws along with suitable initial and boundary conditions are actually sufficient to uniquely specify this function (Lawrence, 2010). Once uniqueness or even well-posedness (Hadamard, 1902) has been established, the question is how to find the function satisfying these laws that will ultimately be the mathematical model. When referring to PDEs in the following, we assume that they have suitable initial and boundary conditions.

In many practical situations, it is impossible to find closed-form solutions and they need to be found numerically. Throughout the last decades, several numerical methods have been proposed and analysed to solve PDEs, especially the *finite element method* (FEM) (Courant, 1943; Hrennikoff, 2021) that may be the standard methodology for a huge class of PDEs. Other techniques are, e.g. the finite difference (Iserles, 2008), finite volume (Eymard *et al.*, 1997) and the spectral element (Patera, 1984) method. These approaches are usually well understood from a theoretical perspective: there are existing error estimators, as well as convergence and stability guarantees. Moreover, the given discretized problems are often in a form that can easily be solved numerically: relying on large, but sparse linear systems or on Newton solvers with good initial guesses and convergence guarantees. While implementing an FEM solver from scratch can be fairly tedious, several multipurpose computational libraries have appeared throughout the years, such as FEniCS (Alnæs *et al.*, 2015) or DUNE (Sander, 2020).

A clear disadvantage of this classical methodology is that it usually relies on a spatial discretization through, e.g. a spatial grid or a large polynomial basis, and thus, lets it suffer from the *curse of dimensionality*: in three space dimensions, it can already be difficult to employ, e.g. the FEM, see, e.g. the discussion in Bungartz & Griebel (2004). In filtering and optimal control, we are easily interested in PDEs occurring in hundreds or thousands of dimensions—here, classical methodology can rarely be employed. In addition, certain nonlinear and non-smooth PDEs are difficult to discretize with, e.g. finite elements due to behaviour that needs to be resolved on a very fine grid, general non-smooth behaviour, or singularities. Since FEM is a mesh-based solver, obtaining quickly converging solutions on certain irregular domains demands tailored approaches that are challenging to design and solve, see, e.g. Egger *et al.* (2014) for domains with re-entrant corners. On irregular domains, the *virtual element method* (Beirão da Veiga *et al.*, 2013) can be more appropriate than FEM.

In recent years, deep learning approaches have become a promising and popular methodology for the numerical solution of various PDEs. They have the potential to overcome some of the challenges that classical methods are facing. That is, neural networks have the advantage to not generally rely on a grid, similar to classical mesh free methods, such as Liu (2009). By leveraging automatic differentiation (Baydin *et al.*, 2018), they eliminate the need for discretization. Additionally, neural networks are able to represent more general functions than, e.g. an FEM basis, and they show evidence of beating the curse of dimensionality, see, e.g. Wojtowytsch & Weinan (2020); Hu *et al.* (2024). While the training of a neural network can become computationally demanding, especially when it consists of a non-convex optimization problem, it is very efficient when evaluating new data points once trained. In this emerging field of deep learning for approximating PDE solutions, the class of approaches closest to the classical methodologies is the one of function approximators (Tanyu *et al.*, 2023). They essentially model the solution as a deep neural network and train the network's parameters to approximate the solution. Such approaches are e.g. the Deep Ritz method (Weinan & Bing, 2018) or the Deep Galerkin method (Sirignano & Spiliopoulos, 2018). A widely used and adapted method of this class are the so-called *physics-informed neural networks* (PINNs) (Raissi *et al.*, 2019) that will be the focus of this paper.

Originally published in a two-part instalment (Raissi *et al.*, 2017a,b), Raissi *et al.* developed the PINNs approach (Raissi *et al.*, 2019)—we will refer to their approach as *vanilla PINNs* throughout the rest of this work. The basic idea behind PINNs is to minimize an energy functional that is the residual of the PDE and its initial and boundary conditions. The neural network itself models the solution function $u(t,x)$ given input variables t and x based on the underlying PDE. It has shown great success for many different types of PDEs (Mao *et al.*, 2020; Smith *et al.*, 2021) and has been extended to various specialized cases (Kharazmi *et al.*, 2019; Jagtap *et al.*, 2020; Yang *et al.*, 2021; Cuomo *et al.*, 2022).

While deep learning approaches for PDEs have gained a lot of traction in the last years and are being employed in increasingly more applications, they come with their set of challenges. In this work, we therefore systematically compare FEM and PINNs in a computational study. Before giving a complete outline of the following, we review recent works on PINNs.

1.1 *PINNs: state of the art*

Compared with FEM, the theoretical groundwork for PINNs is rather sparse. The first work on convergence results with respect to the number of training points is given by Shin *et al.* (Shin *et al.*, 2020). For linear second-order elliptic and parabolic PDEs, they prove strong convergence in C^0 for i.i.d. sampled training data. Mishra *et al.* (Mishra & Molinaro, 2022) have in turn developed upper bounds on the generalization error of PINNs given some stability assumptions on the PDE. Focusing on a specific PDE, Ryck *et al.* (De Ryck *et al.*, 2024) have investigated incompressible Navier–Stokes equations and provided an upper bound on the total error. However, their considerations are limited to the case of networks with two hidden layers and $\tanh$ activation functions. Despite the remaining need for more extensive theoretical work, PINNs have been extended into many different directions. Jagtap *et al.* (Jagtap *et al.*, 2020) develop conservative PINNs (cPINNs) to incorporate complex non-regular geometries by decomposing a spatial domain into independent parts and train separate PINNs for each. This work has been generalized to the extended PINNs (XPINNs) (Jagtap & Karniadakis, 2020) to allow space–time domain decomposition that can be applied to any type of PDE and enables parallelization in training. Another domain decomposition method for PINNs are the hp-VPINNs (Kharazmi *et al.*, 2021). This work is based on the variational PINNs (Kharazmi *et al.*, 2019) that take inspiration from classical approaches for solving PDEs. VPINNs form the loss functional based on the variational form of the PDE with Legendre polynomials as test functions, thus, allowing, e.g. for certain non-smoothnesses in PDEs and also for more efficient training. Bayesian PINNs (Yang *et al.*, 2021) for noisy data employ a Bayesian neural network. Finite basis PINNs (Moseley *et al.*, 2023) combine the PINN-idea with FEM, aiming to reduce the spectral bias in PINNs (Rahaman *et al.*, 2019). Noteworthy is also the work (Fabiani *et al.*, 2021), which uses a neural network with random weights to construct a basis on which PDEs can be solved efficiently. Interesting applications of PINNs appear in fluid dynamics (Mao *et al.*, 2020), electromagnetism (Kovacs *et al.*, 2022), elastic material deformation (Li *et al.*, 2021) and seismology (Smith *et al.*, 2021). For a more extensive review of PINNs, its applications and extended forms, we refer to the survey by Cuomo *et al.* (Cuomo *et al.*, 2022). Some shortcomings of PINNs are documented by Krishnapriyan *et al.* (Krishnapriyan *et al.*, 2021), who find that PINNs struggle to learn relevant physics in more challenging PDE regimes. However, they simultaneously present ideas to address these issues. Other shortcomings of PINNs in computational fluid dynamics were documented by Chuang & Barba (2022).

PINNs were created with physical application in mind. Due to their flexibility, the main building block can be kept the same across many physical problems with smaller adjustments to the architecture. Implementation of the framework has also become more approachable since the release of dedicated software packages such as DeepXDE (Lu *et al.*, 2021b), NVIDIA Modulus (previously SimNet) (Hennigh *et al.*, 2021) or NeuroDiffEq (Chen *et al.*, 2020).

1.2 *Contributions and outline*

As mentioned above, the main goal of this work is a systematic comparison of PINNs and FEM for the solution of PDEs. Indeed, we consider

- the elliptic Poisson equation in one, two and three space-dimensions,
- the parabolic Allen–Cahn equation in one space-dimension and
- the hyperbolic semilinear Schrödinger equation in one and two space dimensions.

We choose these model problems to cover a large range of classes of PDEs. We compare PINNs and FEM in terms of solution time, evaluation time and accuracy. We employ different finite element bases, as well as a multitude of network architectures. Several improvements over the vanilla PINNs have been discussed in the literature. As those are either still fundamentally based on the same idea or rather represent a mix of a classical approach and PINNs, we focus on vanilla PINNs in the following text. Throughout this work, we choose an experimental set-up that is supposed to mimic the way PINNs and FEM are used by practitioners, e.g. computational scientists and engineers, machine learners, mathematical modellers and so on.

This work is structured as follows. We discuss the PDEs, FEM and PINNs in Section 2. We introduce our method of comparison in Section 3, before presenting our computational results regarding Poisson, Allen–Cahn and semilinear Schrödinger equations in Sections 4, 5 and 6, respectively. We discuss our results and conclude the work in Section 7.

2. Mathematical background

In the following text, we first study a very general class of PDEs that we formally denote by

$$\mathscr{A}u(x,t) = f(x,t) \quad x \in \Omega,\ t \in [0,T]. \tag{2.1}$$

Here, the function $u : \overline{\Omega} \times [0,T] \to \mathbb{R}^n$ denotes the solution of the PDE, where $\Omega \subset \mathbb{R}^d$ is open, bounded and connected and usually represents a spatial domain, whereas $[0,T]$ is the time interval. $\mathscr{A}$ denotes the differential operator acting on u that can also be nonlinear and $f : \Omega \times [0,T] \to \mathbb{R}^n$ is a source term. We denote the corresponding boundary conditions and the initial condition by

$$\begin{aligned} \mathscr{B}u(x,t) &= g(x,t), \quad x \in \partial\Omega,\ t \in [0,T] \\ u(x,0) &= h(x), \quad x \in \Omega, \end{aligned} \tag{2.2}$$

respectively.

Throughout this work, we consider three different PDEs: the *Poisson equation*, the *Allen–Cahn equation* and the *semilinear Schrödinger equation*. We introduce those PDEs in the following text.

Poisson equation. The Poisson equation is a linear elliptic PDE of the form

$$\Delta u(x) = f(x), \qquad x \in \Omega, \tag{2.3}$$

where $\Delta = \sum_{i=1}^{d} \frac{\partial^2}{\partial x_i^2}$ denotes the Laplacian and $f : \overline{\Omega} \to \mathbb{R}$ is a source term. To be well defined, we need to equip it with boundary conditions, say, of *Dirichlet*-type:

$$u(x) = u_\partial(x), \qquad x \in \partial\Omega,$$

Neumann-type:

$$\partial_{\vec{n}} u(x) = u_\partial(x) \qquad x \in \partial\Omega,$$

or a combination of the two. In each case, we employ the function $u_\partial : \partial\Omega \to \mathbb{R}$ to model the boundary behaviour. There are several results about the existence of strong and weak solutions of elliptic PDEs, see, e.g. Theorem 3 in Chapter 6.2 in Lawrence (2010). The Poisson equation models, for instance, the stationary heat distribution in a homogeneous object Ω; here, f describes heat sources and sinks.
Allen–Cahn equation. The Allen–Cahn equation is a semilinear parabolic PDE of the form

$$\frac{\partial u(t,x)}{\partial t} = \varepsilon \Delta u(t,x) - \frac{2}{\varepsilon} u(t,x)(1-u(t,x))(1-2u(t,x)), \qquad t \in [0,T],\ x \in \Omega,$$

where $\varepsilon > 0$ and which, of course, is considered together with appropriate boundary conditions and an initial condition. If ε is sufficiently small, the solution of the Allen–Cahn equation approximately partitions the domain Ω into patches where $u \approx 0$ and $u \approx 1$; in-between those patches—at the so-called diffuse interface—it is smooth. These patches can represent binary alloys (Allen & Cahn, 1972) or, e.g. different segments in an image with pixels in Ω (Beneš *et al.*, 2004). The Allen–Cahn equation is also a relaxation of the mean-curvature flow to which it converges as $\varepsilon \downarrow 0$ (Feng & Prohl, 2003).
Semilinear Schrödinger equation. The semilinear Schrödinger equation is a complex-valued nonlinear hyperbolic PDE of the form

$$\mathrm{i}\frac{\partial h(t,x)}{\partial t} = -\Delta h(t,x) - h(t,x)|h(t,x)|^2, \qquad t \in [0,T],\ x \in \Omega,$$

again subject to appropriate initial and boundary conditions. We have represented the semilinear Schrödinger equation above as it is usual in the literature. We present the PDE again splitting real and imaginary parts. We set $h(t,x) =: u_R(t,x) + \mathrm{i}u_I(t,x)$ and write

$$\frac{\partial u_R(t,x)}{\partial t} = -\Delta u_I(t,x) - (u_R^2(t,x) + u_I^2(t,x))u_I(t,x), \qquad t \in [0,T],\ x \in \Omega,$$
$$\frac{\partial u_I(t,x)}{\partial t} = \Delta u_R(t,x) + (u_R^2(t,x) + u_I^2(t,x))u_R(t,x), \qquad t \in [0,T],\ x \in \Omega.$$

We refer to Rozenman *et al.* (2020) and Strauss (1978) for an application of the semilinear Schrödinger equation in fluid dynamics and in nonlinear optics, respectively.

2.1 *Finite element method*

As mentioned before, FEM has been the gold standard for the spatial discretization of a huge class of PDEs. We now discuss the basics of FEM given the example of an elliptic PDE as in (2.3) with homogeneous Dirichlet boundary $u_\partial = 0$ and square integrable source $f \in \mathscr{L}^2(\Omega)$. We mention non-stationary PDEs and other boundary conditions further below.

When solving an elliptic PDE with FEM, we aim to find a weak solution. That means, we try to find u in an appropriate function space U such that for all functions v in an appropriate function space V, we have

$$\int_\Omega v(x)(\Delta u(x) - f(x))\mathrm{d}x = 0.$$

Applying integration by parts, we obtain the usual weak formulation of the Poisson equation:

$$\int_\Omega \langle \nabla v(x), \nabla u(x)\rangle + v(x)f(x)\mathrm{d}x = 0. \tag{2.4}$$

An appropriate choice of function spaces U and V in this case is $U = V = H_0^1(\Omega)$, the Sobolev space of square-integrable functions $\Omega \to \mathbb{R}$ that are zero at the boundary $\partial\Omega$ and that have a square-integrable weak derivative. In finite elements, we now replace $H_0^1(\Omega)$ by a finite-dimensional subspace $H \subseteq U$ with basis $(\varphi_i)_{i=1}^N$. On this finite-dimensional space, we can write the weak form (2.4) as

$$-\int_\Omega \langle \nabla\varphi_i(x), \nabla\left(\sum_{j=1}^N a_j\varphi_j(x)\right)\rangle\mathrm{d}x = \int_\Omega \varphi_i(x)f(x)\mathrm{d}x \qquad i = 1, ..., N, \tag{2.5}$$

which can be rewritten as a usual linear system $Ba = b$, with

$$B = \left(-\int_\Omega \langle\nabla\varphi_i(x), \nabla\varphi_j(x)\rangle\mathrm{d}x\right)_{i,j=1}^N, \qquad b = \left(\int_\Omega \varphi_i(x)f(x)\mathrm{d}x\right)_{i=1}^N.$$

Now, we use $\sum_{j=1}^N a_j\varphi_j$ to approximate the solution u of the PDE and obtain convergence to u when appropriately choosing $(\varphi_i)_{i=1}^N$, e.g. (Iserles, 2008, Theorem 9.3). While this approach allows for a large range of such bases, we usually speak only of finite elements when choosing specific locally supported, piecewise polynomial functions. As usual differential operators are local, this will usually lead to B having a favourable, sparse structure.

In the discussion above, we consider homogeneous Dirichlet boundary conditions $u_\partial = 0$. Inhomogeneous boundary conditions, in which u_∂ is not constantly 0, can be achieved by first finding a function that satisfies the boundary conditions and then solving an auxiliary homogeneous problem with $u_\partial = 0$. The weak formulation for Neumann conditions adds the additional term $\int_{\partial\Omega} vu_\partial \mathrm{d}x$ to the bilinear form and is solved on a different function space $U = V = H^1(\Omega)$. Indeed, this is the Sobolev space of square-integrable functions with square-integrable weak derivatives. Mixed boundary conditions can be enforced in a similar way.

The time-domain of a non-stationary PDE can also be discretized with finite elements, see, e.g. Hulbert & Hughes (1990). We, however, consider the *method of lines* using finite elements in space and usual integrators for initial value problems in time, see, e.g. Schiesser & Griffiths (2009). To represent

Allen–Cahn and semilinear Schrödinger at the same time, we consider some semilinear PDE of the form

$$\frac{\partial u(t,x)}{\partial t} = \Delta u(t,x) + F(u(t,x)), \quad u(0,x) = u_0(x) \qquad t \in [0,T],\, x \in \Omega,$$

which is also subject to boundary conditions. Due to the stiffness of the heat equation, we are required to use implicit techniques, such as the implicit Euler method. In a semi-discrete form, we can write the update step as

$$u_{k+1} = u_k + \delta t \Delta u_{k+1} + \delta t F(u_{k+1}) \qquad k = 0, 1, \dots,$$

where $\delta t > 0$ denotes the size of the time steps and $u_k(x) = u(t_k, x)$ for $k = 0, 1, \dots$. In the weak formulation, we obtain

$$\int_\Omega v u_{k+1} \mathrm{d}x = \int_\Omega v u_k - \delta t \langle \nabla v, \nabla u_{k+1} \rangle + \delta t v F(u_{k+1}) \mathrm{d}x \qquad v \in V, k = 0, 1, \dots$$

and can now again use a finite element discretization in space to obtain an approximation of u_{k+1} represented through a finite-dimensional equation, similar to (2.5), with the $(a_j)_{j=1}^N$ now depending on time. In the case of Allen–Cahn and semilinear Schrödinger, this equation is nonlinear and requires us to repeatedly employ a Newton's method.

We can prevent the additional cost of a Newton-solve and usually still obtain a stable discretization, by employing a semi-implicit strategy: linear parts of the PDE are solved with an implicit discretization, while nonlinear parts are discretized explicitly. In our setting, we obtain

$$u_{k+1} = u_k + \delta t \Delta u_{k+1} + \delta t F(u_k) \qquad k = 0, 1, \dots$$

which requires us to solve the following weak problem:

$$\int_\Omega v u_{k+1} + \delta t \langle \nabla v, \nabla u_{k+1} \rangle \mathrm{d}x = \int_\Omega v u_k + \delta t v F(u_k) \mathrm{d}x \qquad v \in V,\, k = 0, 1, \dots$$

which is fully linear.

For more details on FEM, we refer to, e.g. the book by Braess (Braess, 2007) or the classical references Courant (Courant, 1943) and Hrennikoff (Hrennikoff, 2021). Integrators for initial value problems, as well as evolution equations, are thoroughly considered by Iserles (Iserles, 2008). Semi-implicit schemes are thoroughly discussed by Koto (2008) and appear, for instance, in the work by Bertozzi and Schönlieb (Bertozzi & Schönlieb, 2010).

2.2 *Physics-informed neural networks*

The aim of PINNs is to approximate PDE solutions using a deep neural network. They use the powerful tool that is automatic differentiation and therefore do not rely on discretization of the space-time domain but rather on random sampling of the domain.

Let us consider a PDE of the form (2.1) with suitable boundary and initial conditions (2.2). The vanilla PINNs approach, as introduced by Raissi *et al.* (Raissi *et al.*, 2019), uses a fully connected neural network with N_l hidden layers and $N_e(l)$ neurons per layer l. Inputs of the network are the independent variables (x,t) sampled in the domain $\Omega \times [0,T]$. The neural network acts as the function approximator $u_\theta(x,t)$ of the PDE solution, with θ the network weights that are optimized during training. Although FEM approximates the PDE solution on a space spanned by a basis of finite elements, PINNs approximate PDEs on the so-called Barron spaces—those were studied in Ma & Lei (2022). The PDE is integrated as a soft constraint in the optimization. That is, the network is trained with the PDE residual, as well as boundary and initial condition residuals as the loss functional:

$$Loss(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}\|\mathscr{A}u_\theta(x_f^i, t_f^i) - f(x_f^i, t_f^i))\|^2 \tag{2.6}$$

$$+\frac{1}{N_g}\sum_{j=1}^{N_g}\|\mathscr{B}u_\theta(x_g^j, t_g^j) - g(x_g^j, t_g^j)\|^2 + \frac{1}{N_h}\sum_{k=1}^{N_h}\|u_\theta(x_h^k, 0) - h(x_h^k)\|^2.$$

Here, N_f is the number of collocation points $(x_f^i, t_f^i) \in \Omega \times [0, T]$ for $i = 1, \ldots, N_f$ sampled for the PDE residual in the loss. Similarly, $(x_g^j, t_g^j) \in \partial\Omega \times [0, T]$ for $j = 1, \ldots, N_g$ denote the training points on the boundary and $x_k^h \in \Omega$ for $k = 1, \ldots, N_h$ the training data for the initial condition. Additionally, we need data $g(x_g^j, t_g^j)$ for the boundary and $h(x_h^k)$ initial conditions. This can be measured data and does not need to be represented as an analytic function. PINNs is therefore able to incorporate measurements and leverage data-driven information. In the following text, we will only be using the 2-norm for the loss; however, this can be amended depending on the problem. The choice of N_f, N_g and N_h is dependent on the domain and complexity of the problem determined heuristically in our case. We follow Raissi *et al.* (2019) in using Latin Hybercube Sampling (Stein, 1987), a quasi-random approach for space filling sampling, to obtain the collocation points for training. In our experiments, we re-sample the collocation points in every epoch to get a better coverage of the sampling domain; see also Jin *et al.* (2023). As mentioned above, the differential operator $\mathscr{A}$ and any derivatives in the boundary condition operator $\mathscr{B}$ are evaluated using automatic differentiation (Baydin *et al.*, 2018). In contrast to numerical methods such as finite differences, automatic differentiation uses the chain rule to backpropagate through the network and evaluate the derivative. It is therefore not dependent on a grid or mesh that discretizes the domain. In turn, the sampling method becomes more important. Automatic differentiation gives rise to a significant advantage of PINNs towards other classical numerical methods for solving PDEs; it does not need to resolve derivatives on small grids, as in finite differences, and scales well in higher dimensions. In PINNs, the design of the neural network architecture, i.e. the type of network structure, the number of hidden layers and the number of nodes per layer, is flexible and can be adjusted based on the PDE complexity. In the following text, we will use fully connected feed-forward dense neural network. We denote the size and numbers of nodes per layers as $[N_e(1), N_e(2), ..., N_e(l)]$. That is, $[5, 5, 1]$ e.g. represents a neural network with two hidden layers and five nodes per layer. The last entry 1 represents the size of the output layer. Lastly, the loss function is typically minimized using first the Adam optimizer (Kingma & Ba, 2015) for a coarser optimization and then the second-order quasi-Newton optimizer L-BFGS (Liu & Nocedal, 1989) which we found improved the accuracy.

3. Method of comparison

In this section, we give an overview of the experimental design and the measures that we consider to compare FEM and PINNs. The main features that we investigate are the time to solve the PDEs and the accuracy of the results. We therefore ask the questions of which methodology is computationally faster, more accurate and most efficient. We investigate different types of PDEs with varying complexity and dimensionality to cover a broad spectrum of PDEs and examine if computational speed and accuracy of the two approaches change based on the PDE type.

3.1 *Experimental set-up*

We now discuss the experimental set-up of our computational study. We first discuss ground truth solutions, and then describe the set-up for FEM and PINNs and the metrics with which we compare them.

Indeed, for a systematic comparison of FEM and PINNs, we need a ground truth solution of the PDEs to compare the solution approximations and to evaluate their accuracy. The first step is therefore to determine these ground truth solutions. For the Poisson equations, we have analytical solutions that we can use to determine the accuracy of the approximation approaches. However, neither the Allen–Cahn equation nor the semilinear Schrödinger equation have analytical solutions. Instead, we use FEM on a very fine mesh to compute a very accurate reference solution that we will use as the ground truth; time-stepping is performed using the implicit Euler method. One of the advantages of FEM is that we have extensive theoretical foundations for the convergence for large classes of PDEs, including the Allen–Cahn equation, e.g. Feng & Hai-jun (2005) and the semilinear Schrödinger equation, e.g. Shi *et al.* (2016). Thus, a finely meshed finite element solution can therefore guarantee accuracy up to a small error.

Let us now discuss the details and specifications for FEM that we consider in the comparison. As mentioned before, finer meshes will lead to higher accuracy albeit slower computation time. We therefore solve the PDEs for different mesh sizes to see the relationship between computation time and accuracy based on the FEM design. For time-dependent PDEs, the Allen–Cahn and the semilinear Schrödinger equations, we choose a semi-implicit strategy for discretization as detailed in Subsection 2.1. All FEM solution approximations are implemented using the Python toolbox FEniCS (version 2019.1.0) (Logg *et al.*, 2012; Alnæs *et al.*, 2015).

In the case of PINNs, we closely follow the vanilla approach as described in Raissi *et al.* (Raissi *et al.*, 2019). That is, we design the loss functional (2.6) and choose weights in the Allen–Cahn loss heuristically, elsewhere they are equal to 1. We use the Adam optimizer (Kingma & Ba, 2015) in the first instance and L-BFGS (Liu & Nocedal, 1989) to refine the optimization. The combination of the two optimizers allows for a fast coarse optimization followed by a computationally expensive refinement. We observed this to improve accuracy. The learning rate is heuristically chosen for optimal results in each PDE. All derivatives in the loss functional are computed using automatic differentiation. The collocation points that make up the input to the neural network are newly sampled for each epoch using Latin Hybercube sampling (Stein, 1987) in the Adam optimization. This way, we are able to cover more of the sampling space and improve generalizability of the trained network. L-BFGS does not allow for re-sampling between iterations and we therefore only sample the collocation points once before the optimization again using Latin Hybercube sampling. The number of collocation points were chosen heuristically and we do not use mini-batching in Adam. PINNs allow for a flexible design of the neural network architecture based on the underlying PDE. We therefore run the network training on architectures of different sizes, i.e. with varying numbers of layers and nodes. The code for PINN training and evaluation was written with the Python neural network library jax (Bradbury *et al.*, 2018) and is available on github: https://github.com/TamaraGrossmann/FEM-vs-PINNs.

We consider three main features for comparison. That is, we evaluate the FEM and PINNs approaches based on their solution time, evaluation time and accuracy. The solution time refers to the time it takes for each method to approximate the general solution. We differentiate between solution and evaluation time due to the way FEM and PINNs approximate solutions differently. FEM, typically, approximates the PDE solution on a fixed mesh. The solution can be approximated beyond the mesh by evaluation of the basis functions and/or interpolation—we always use linear interpolation in time. On the other hand, for PINNs a neural network is trained on a set of collocation points. The trained network can then be evaluated at any point in the domain. While the training time of PINNs can be rather slow, one of the advantages of neural networks is that once trained the evaluation on new input data require only a forward evaluation of the neural network, which is very fast. We therefore consider both times. For FEM this means solution time refers to assembling the matrix and solving the weak form of the PDE on a fixed mesh. FEM is run on a CPU. Considering PINNs, the solution time refers to the training time of the neural network which is run on a GPU. In turn, the evaluation time in FEM is measured as the time to interpolate the solution on a different mesh. In PINNs, the evaluation time is the time to evaluate the trained neural network on a new set of collocation points. We measure the accuracy of both methods on the same mesh in comparison with the ground truth solutions. For the Allen–Cahn and Schrödinger equations the evaluation mesh is chosen as the fine mesh on which the ground truth solution was derived. The measure for accuracy is the ℓ^2 relative error. All experiments were run on two machines with identical specifications: 12 CPU cores (Intel® Xeon® CPU E5-2643 v3 @ 3.40GHz) and a single Nvidia Quadro P6000 GPU with a base clock speed of 1506MHz. The codes were run 10 times and the reported solution and evaluation times are the average over the 10 recorded times.

It is not easy to find an experimental set-up that is appropriate to compare PINNs and FEM. With the set-up explained above, we aim to compare PINNs and FEM in the way they are intended to use by applicants. The FEM can sometimes be significantly sped-up using a GPU acceleration, see, e.g. Kiran *et al.* (2023). However, the standard mode of operation for the FEM appears to be based on CPUs. Similarly, modern automatic differentiation libraries are based on GPU-accelerated backpropagation, see, e.g. Sierra-Canto *et al.* (2010). We would assume a much slower training time of the PINNs, if we applied the backpropagation purely CPU-based. Thus, the way of comparison we use is appropriate in the sense that it represents the usual way in which PINNs and FEM is used by applicants.

4. Approximating the Poisson equation

Let us first investigate the Poisson equation in one, two and three space-dimensions. Each of the equations we consider has an analytical solution that we can use to evaluate the accuracy of the FEM and PINN approximation. Comparing the same type of PDE in different dimensions also allows us to draw conclusions about cost and accuracy effects due to the dimensionality of the PDE.

4.1 *1D*

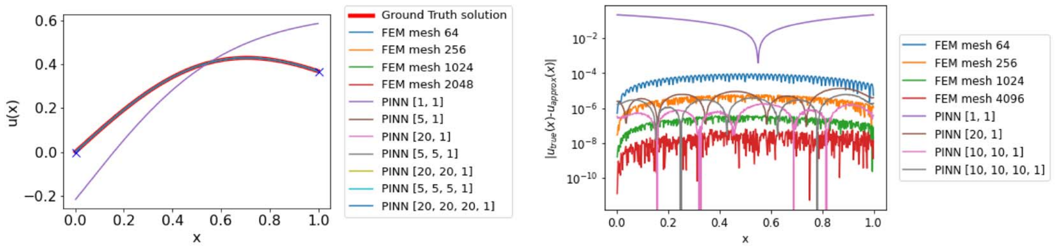
For the one-dimensional case, we are examining the Poisson equation with a right-hand side $f(x)$ as detailed below on a unit interval and employ Dirichlet boundary conditions:

$$\begin{aligned} \Delta u(x) &= (4x^3 - 6x)\exp(-x^2), \quad x \in (0,1), \\ u(0) &= 0, \\ u(1) &= \exp(-1). \end{aligned} \tag{4.1}$$

This PDE has the analytical solution

$$u_{\text{true}}(x) = x\exp(-x^2), \qquad x \in [0,1].$$

A visualization of the solution to the 1D Poisson equation is shown in Fig. 1(a).



(a) Plot of the PDE ground truth solution and some of the FEM and PINNs solution approximations.

(b) Plot of the difference between some of the FEM / PINNs solution approximations to the ground truth solution on a log scale.

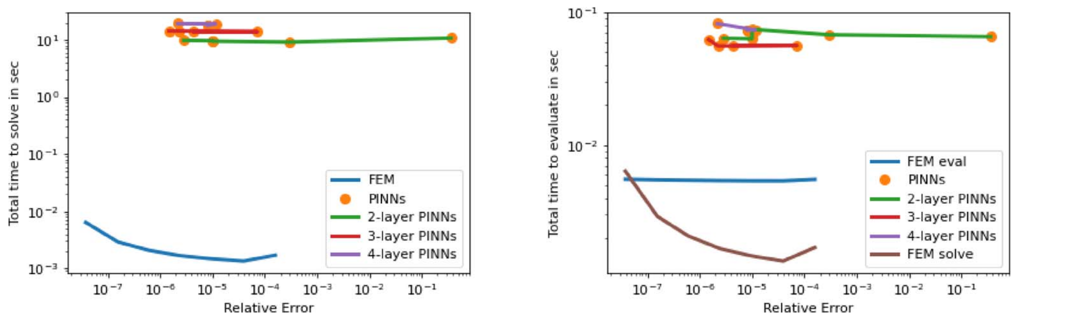
FIG. 1. Plot for 1D Poisson equation solution.

4.1.1 *FEM.* As introduced in Section 2.1, the first step in solving (4.1) with FEM is deriving the weak formulation of the PDE, which we have done for Poisson equation already in (2.4). Here, of course $f(x) = (4x^3 - 6x)\exp(-x^2)$. Next, we need to define the finite element mesh. We choose a regular mesh on the domain $[0,1]$ with varying numbers of cells $n \in \{64, 128, 256, 512, 1024, 2048, 4096\}$. Of course, a higher number of cells corresponds to a finer grid on which the PDE is solved and subsequently leads to a more accurate, but also computationally more costly solution. The finite elements we are using are standard linear Lagrange elements, i.e. piecewise linear hat functions (P_1). Subsequently, we solve the variational problem in (2.4) with the Dirichlet boundary conditions specified in (4.1) using the conjugate gradient method with incomplete LU factorization as a preconditioner. All specifications are implemented using the python toolbox FEniCS (Logg *et al.*, 2012; Alnæs *et al.*, 2015) to compute an approximate PDE solution.

4.1.2 *PINNs.* For solving the 1D Poisson equation using PINNs, there are three design parameters that we need to specify before training. The first step is choosing a loss functional. Following the vanilla PINNs approach, we evaluate the goodness of the solution using the discretized ℓ^2-energy or mean squared error over the PDE, boundary and initial conditions. In particular, we define the loss function as

$$\text{Loss}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|\Delta u_\theta(x_i) - (4x_i^3 - 6x_i)\exp(-x_i^2)\|_2^2 + \|u_\theta(0)\|_2^2 + \|u_\theta(1) - \exp(-1)\|_2^2,$$

with u_θ the neural network, θ the trained weights and $N = 256$ the number of collocation points x_i sampled in each epoch using latin hybercube sampling. The second design parameter is the neural network architecture, i.e. the type of neural network, the activation function and the number of hidden layers and nodes. For the 1D Poisson case, we train feed-forward dense neural networks with $\tanh$ as the activation function. We compare the results on architectures of different sizes. The network architectures we consider for 1D Poisson are $[1,1], [2,1], [5,1], [10,1], [20,1], [40,1], [5,5,1], [10,10,1], [20,20,1], [40,40,1], [5,5,5,1], [10,10,10,1], [20,20,20,1]$ and $[40,40,40,1]$. The loss function is minimized for each network architecture using the Adam optimizer for 15,000 epochs with a learning rate of 10^{-4} in the first instance. Additionally, we refine the optimization using L-BFGS.



(a) Plot of time to solve FEM and train PINN in sec versus the ℓ^2 relative error.

(b) Plot of time to interpolate FEM and evaluate PINN in sec versus the ℓ^2 relative error.

FIG. 2. Plot for 1D Poisson equation of time in sec versus the ℓ^2 relative error.

4.1.3 *Results.* The resulting solution approximations of the 1D Poisson equation using FEM and PINNs are compared with the analytical solution on a mesh in $[0, 1]$ with 512 mesh points. The ground truth solution of the 1D Poisson equation in (4.1) and its approximations are displayed in Fig. 1(a). Likewise, the difference from the approximate solutions to the GT solution is shown in Fig. 1(b) for all mesh sizes of FEM and architectures in PINNs. One of the architectures in the PINNs approximation renders results with large relative error: the PINN with a single hidden layer and one node is not even able to learn the solution in a way in which the boundary conditions are satisfied. However, all other solution approximations differ from the ground truth only marginally.

Let us compare the time it takes to solve or rather approximate the PDE using FEM and PINNs to the relative error produced on a new set of grid points. For FEM the solution time is the time to solve the linear systems and for PINNs we consider the time to train the neural network. The results are shown in Fig. 2(a). We can clearly show that overall FEM is faster and more accurate in their solution approximation. While there are some PINN architectures that are able to achieve similar or even lower relative errors than the coarser FEM approximations, their training time is two to three orders of magnitude higher than in FEM. Considering the evaluation time, i.e. the time to interpolate the FEM solution on a different mesh and evaluate the trained PINN on a test set, we can see a similar relationship; see Fig. 2(b). FEM solution approximations are overall faster and more accurate. Finally, we consider the relationship between the number of layers and the solution time and relative error. We observe that the time to train a PINN is similar relative to the number of layers. However, the accuracy of each network is related to the number of nodes in the layers. That is, we cannot show that networks with more layers generally achieve better results in 1D Poisson.

4.2 *2D*

Let us now investigate a two-dimensional Poisson equations given by

$$\begin{aligned} \Delta u(x,y) = {} & 2(x^4(3y-2) + x^3(4-6y) + x^2(6y^3 - 12y^2 + 9y - 2) \\ & - 6x(y-1)^2 y + (y-1)^2 y) \qquad (x,y) \in (0,1)^2 \end{aligned} \tag{4.2}$$

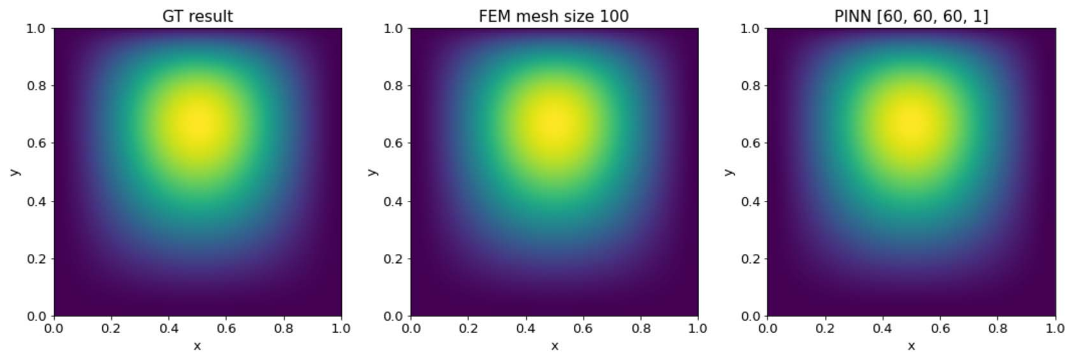


FIG. 3. Comparison of the 2D Poisson ground truth solution to examples of the FEM and PINN approximations.

$$\begin{aligned} \partial_{\vec{n}} u(0,y) &= 0 \quad y \in [0,1] \\ \partial_{\vec{n}} u(1,y) &= 0 \quad y \in [0,1] \\ u(x,0) &= 0 \quad x \in [0,1] \\ \partial_{\vec{n}} u(x,1) &= 0 \quad x \in [0,1] \end{aligned} \tag{4.3}$$

We can solve the PDE (4.2) with mixed boundary conditions analytically:

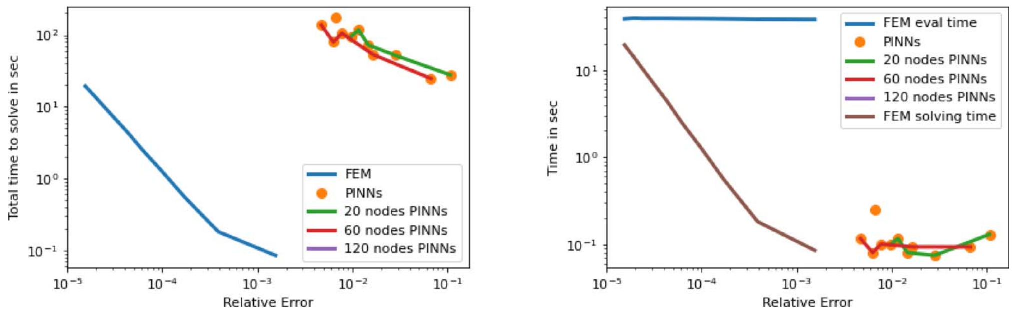
$$u_{\text{true}}(x,y) = x^2(x-1)^2 y(y-1)^2, \qquad (x,y) \in (0,1)^2.$$

The solution is displayed in Fig. 3.

4.2.1 *FEM.* For the 2D Poisson equation, we again refer to the weak formulation of the Poisson equation in (2.4) with $f(x,y) = 2(x^4(3y-2) + x^3(4-6y) + x^2(6y^3 - 12y^2 + 9y - 2) - 6x(y-1)^2 y + (y-1)^2 y)$. The finite element mesh is defined on the unit square $[0,1] \times [0,1]$; it contains $n \times n$ squares with $n \in \{100, 200, \dots, 1000\}$. Each square on the mesh is divided into two triangles on which we define again piecewise linear finite elements (P_1). We solve the variational problem (2.4) with mixed boundary conditions given in (4.3) using the conjugate gradient method with incomplete LU factorization preconditioning. The method is implemented using FEniCS.

4.2.2 *PINNs.* The loss functional for the PINN approximation is the ℓ^2-residual of the PDE (4.2) and its boundary conditions. For u_θ the neural network with weights θ that are to be trained, the loss reads:

$$\begin{aligned} &\text{Loss}(\theta) \\ &= \frac{1}{N_f} \sum_{i=1}^{N_f} \| \Delta u_\theta(x_f^i, y_f^i) - 2((x_f^i)^4(3y_f^i - 2) + (x_f^i)^3(4 - 6y_f^i) + (x_f^i)^2(6(y_f^i)^3 - 12(y_f^i)^2 + 9y_f^i - 2) \|_2^2 \\ &\quad + \frac{1}{N_g} \sum_{j=1}^{N_g} \left(\left\| \partial_{\vec{n}} u_\theta(0, y_g^j) \right\|_2^2 + \left\| \partial_{\vec{n}} u_\theta(1, y_g^j) \right\|_2^2 + \left\| u_\theta(x_g^j, 0) \right\|_2^2 + \left\| \partial_{\vec{n}} u_\theta(x_g^j, 1) \right\|_2^2 \right). \end{aligned}$$



(a) Plot of time to solve FEM and train PINN in sec versus the ℓ^2 relative error.

(b) Plot of time to interpolate FEM and evaluate PINN in sec versus the relative error. For comparison, the time to solve FEM is also plotted.

FIG. 4. Plot for 2D Poisson equation of time in sec versus the ℓ^2 relative error.

The collocation points are sampled at every epoch using Latin Hybercube sampling with $N_f = 2000$ and $N_g = 250$. We train a feed-forward dense neural network with tanh activation function. We consider 11 different architectures for training, these are [20,1], [60,1], [20,20,1], [60,60,1], [20,20,20,1], [60,60,60,1], [20,20,20,20,1], [60,60,60,60,1], [20,20,20,20,20,1], [60,60,60,60,60,1] and [120,120,120,120,120,1]. Just like in 1D Poisson, we use the Adam optimizer for $20,000$ epochs with a learning rate of 10^{-3} to train the network. Subsequently, we refine the optimization using L-BFGS.

4.2.3 *Results.* An example of the resulting solution approximations on a mesh with 2000×2000 cells next to the ground truth solution is show in Fig. 3. The time versus accuracy plots for the 2D Poisson equation are displayed in Fig. 4(a) and Fig. 4(b). Considering the solution time, FEM clearly outperforms all PINN approximations both in accuracy and computation time. Using FEM to obtain the PDE solution is faster by one to three orders of magnitude. However, when we look at the evaluation times in Fig. 4(b), the relationship changes. That is, the time to evaluate a PINN is two to three orders of magnitude faster than the interpolation of FEM on a new mesh. We additionally plotted the FEM solution time in the same plot, and the PINN evaluation remains faster, however, by a smaller margin. Interestingly, the solution time of FEM is faster than the FEM evaluation time; this is possibly due to an inefficient interpolation code. Even though the evaluation of the trained neural network gives an improvement in time, the PINN approximations remain to have lower accuracy.

4.3 *3D*

For the three-dimensional Poisson equation, we choose a PDE on the unit cube with Dirichlet boundary conditions, as follows:

$$\begin{aligned}\Delta u(x,y,z) &= -3\pi^2 \sin(\pi x)\sin(\pi y)\sin(\pi z), \quad (x,y,z) \in (0,1)^3,\\ u(x,y,z) &= 0, \quad (x,y,z) \in \partial(0,1)^3.\end{aligned}$$

The analytical solution of this 3D Poisson equation is displayed in Fig. 5 and can be written as

$$u_{\text{true}}(x,y,z) = \sin(\pi x)\sin(\pi y)\sin(\pi z), \quad (x,y,z) \in (0,1)^3.$$

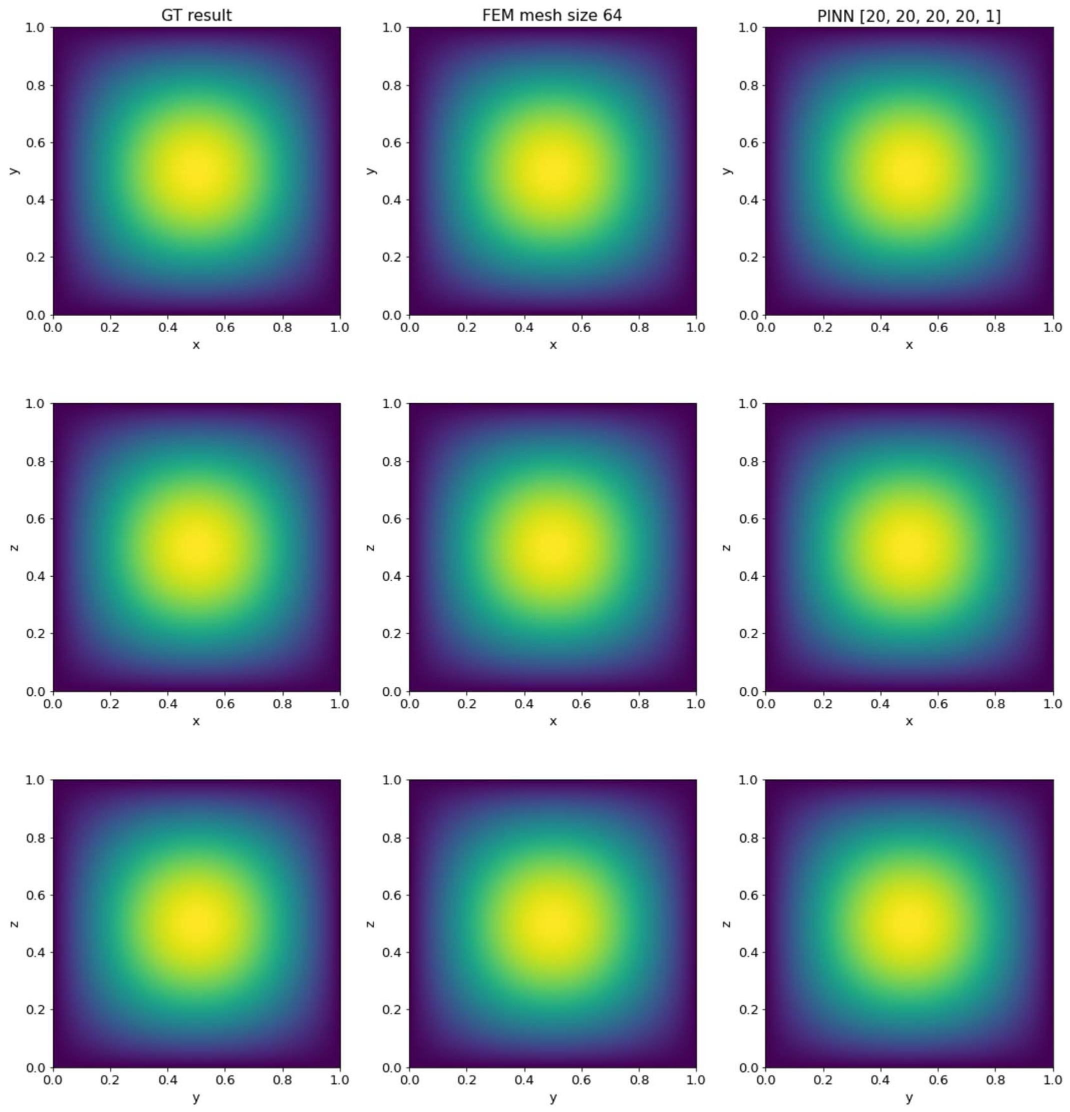


FIG. 5. Comparison of the 3D Poisson ground truth solution slices at $x, y, z = 0.5$, respectively, to examples of the FEM and PINN approximations.

4.3.1 *FEM.* The weak formulation of the 3D Poisson equation is again given in (2.4) with $f(x, y, z) = -3\pi^2 \sin(\pi x) \sin(\pi y) \sin(\pi z)$. The finite element mesh is defined on a unit cube $[0, 1] \times [0, 1] \times [0, 1]$ consisting of $n \times n \times n$ cubes with $n \in \{16, 32, 64, 128\}$. We subdivide each cube into tetrahedrals and use again piecewise linear finite elements (P_1). The weak problem is solved using the conjugate gradient method and incomplete LU factorization as the preconditioner.

4.3.2 *PINNs.* Similar to the one- and two-dimensional Poisson case, we design the loss functional as the PDE and boundary condition residual. That is, we define the loss as follows:

$$
\begin{aligned}
\mathrm{Loss}(\theta) = {} & \frac{1}{N_f}\sum_{i=1}^{N_f} \| \Delta u_\theta(x_f^i, y_f^i, z_f^i) + 3\pi^2 \sin(\pi x_f^i)\sin(\pi y_f^i)\sin(\pi z_f^i)\|_2^2 \\
& + \frac{1}{N_g}\sum_{j=1}^{N_g}\left(\|u_\theta(0, y_g^j, z_g^j)\|_2^2 + \|u_\theta(x_g^j, 0, z_g^j)\|_2^2 + \|u_\theta(x_g^j, y_g^j, 0)\|_2^2\right) \\
& + \frac{1}{N_g}\sum_{j=1}^{N_g}\left(\|u_\theta(1, y_g^j, z_g^j)\|_2^2 + \|u_\theta(x_g^j, 1, z_g^j)\|_2^2 + \|u_\theta(x_g^j, y_g^j, 1)\|_2^2\right).
\end{aligned}
$$

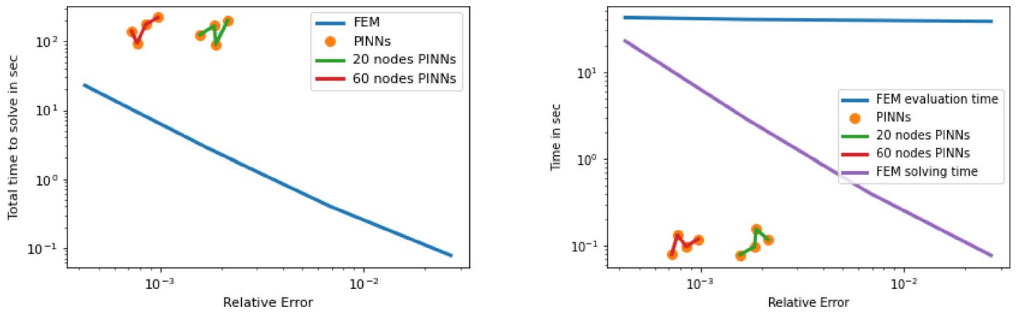
Here, u_θ denotes the neural network with θ the training parameters. We sample $N_f = 1000$ collocation points in the domain and $N_g = 100$ on the boundary. The neural network is again a feed-forward dense neural network with $\tanh$ activation function. We consider eight different architectures to train with the following layer and node specifications: [20,20,1], [60,60,1], [20,20,20,1], [60,60,60,1], [20,20,20,20,1], [60,60,60,60,1], [20,20,20,20,20,1] and [60,60,60,60,60,1]. The Adam optimizer with learning rate 10^{-3} is used for 20,000 epochs before employing the L-BFGS optimizer.

4.3.3 *Results.* The PDE is evaluated on a mesh of $150 \times 150 \times 150$ grid points and example results are shown in Fig. 5. The time versus accuracy plots can be found in Fig. 6. As expected, the FEM results with slower computation times have lower relative errors as they are solved on finer meshes as shown in Fig. 6(a). While the PINN approximations are about one to three orders of magnitude slower in training time depending on the FEM mesh solution that we compare with, they are able to achieve equal and even higher accuracy in most cases. On the other hand, PINNs outperform FEM when considering the evaluation time as plotted in Fig. 6(b). The time to interpolate FEM on a new mesh is two to three orders of magnitude slower than the evaluation time of PINNs. Additionally, PINNs are able to achieve equal or, in the case of coarse FEM approximations, higher accuracy scores. If we also take the FEM solving time into account, we find again that this is faster than the evaluation time. However, it is slower than the PINNs evaluation. Therefore, a trained PINN has a lower computation time for the evaluation as compared with both the FEM solution and evaluation times.

5. Approximating the Allen–Cahn equation

To study the one dimensional Allen–Cahn equation, we consider the following PDE:

$$
\begin{aligned}
\frac{\partial u(t,x)}{\partial t} &= \epsilon \Delta u - \frac{2}{\epsilon} u(t,x)(1-u(t,x))(1-2u(t,x)), \quad x \in \Omega = [0,1],\ t \in [0,T], \\
u(t,0) &= u(t,1), \quad t \in [0,T], \qquad (5.1)\\
u(0,x) &= \frac{1}{4}(\sin(2\pi x) + \sin(16\pi x)) + \frac{1}{2}, \quad x \in \Omega,
\end{aligned}
$$



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.

(b) Plot of time to interpolate FEM and evaluate PINN in sec versus relative error. For comparison, the time to solve FEM is also plotted.

FIG. 6. Plot for 3D Poisson equation of time in sec versus ℓ^2 relative error.

where $T = 0.05$ and $\epsilon = 0.01$. As mentioned in Section 2, we note that a smaller ϵ will yield close to piecewise constant solutions, whereas solutions with large ϵ will be overall more smooth. The ϵ chosen in this case is due to the inability of PINNs to approximate the Allen–Cahn solution with a smaller ϵ. We have trained the neural network to solve Allen–Cahn with $\epsilon = 0.001$ for various network architectures and with activation functions such as softplus that are typically able to handle discontinuous solutions. However, the PINNs were not able to recover results close to the ground truth solution. We will discuss this case further below. For now, the results refer to $\epsilon = 0.01$.

5.0.1 *FEM*

As opposed to the Poisson equation, the Allen–Cahn equation has a time dependency that we need to consider in the discretization. As mentioned in Section 2.1, we use an implicit Euler strategy to approximate the solution. The weak formulation in a semi-discrete form of the 1D Allen–Cahn equation with homogeneous Dirichlet boundary $u_\partial = 0$ can be written as

$$\int_0^1 (u_{k+1}(x) - u_k(x))v(x)\mathrm{d}x + \epsilon \int_0^1 \langle \nabla u_{k+1}(x), \nabla v(x)\rangle \mathrm{d}x \\ + \frac{2}{\epsilon}\int_0^1 u_{k+1}(x)(1-u_{k+1}(x))(1-2u_{k+1}(x))v(x)\mathrm{d}x = 0, \tag{5.2}$$

for all test functions $v \in H_0^1([0,1])$. The time is discretized with distance of $dt = 10^{-3}$. For the finite element mesh in space, we choose an interval mesh on the domain $[0,1]$ with varying numbers of cells $n \in \{32, 128, 512, 2048\}$. As for the 1D Poisson problem, we employ piecewise-linear finite element functions (P_1). The nonlinear variational problem (5.2) with periodic boundary conditions is solved using a Newton solver. The FEM approach was implemented using the Python toolbox FEniCS.

5.0.2 *PINNs*

We follow the general PINNs approach and design the loss based on the PDE residual, the boundary residual and the initial condition residual. Additionally, we introduce a weighting of the different terms in the loss functional. We found heuristically this weighting to render the best results.

$$\begin{aligned}\mathrm{Loss}(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}\left\| \frac{\partial u_\theta(t_f^i, x_f^i)}{\partial t} - \epsilon \Delta u_\theta(t_f^i, x_f^i) + \frac{2}{\epsilon} u_\theta(t_f^i, x_f^i)\big(1 - u_\theta(t_f^i, x_f^i)\big)\big(1 - 2u_\theta(t_f^i, x_f^i)\big)\right\|_2^2 \\ + \frac{1}{N_g}\sum_{k=1}^{N_g} \| u_\theta(t_g^j, 0) - u_\theta(t_g^j, 1)\|_2^2 \\ + \frac{1000}{N_h}\sum_{k=1}^{N_h}\left\| u_\theta(0, x_h^k) - \frac{1}{4}\left(\sin(2\pi x_h^k) + \sin(16\pi x_h^k)\right) + \frac{1}{2}\right\|_2^2,\end{aligned} \tag{5.3}$$

with u_θ the neural network and θ the trained weights. We train the network on $N_f = 20,000$ collocation points $(t_f^i, x_f^i) \in [0, 0.05] \times [0, 1]$ that are sampled using Latin Hybercubes. The training points for the boundary with $N_g = 250$ and for the initial condition with $N_h = 500$ are also sampled in each epoch with Latin Hybercube sampling. The network architecture is a feed-forward dense neural network with a tanh activation function. We consider architectures of the following 14 different sizes: [20,20,20,1], [100,100,100,1], [500,500,500,1], [20,20,20,20,1], [100,100,100,100,1], [500,500,500,500,1], [20,20,20,20,20,1], [100,100,100,100,100,1], [500,500,500,500,500,1], [20,20,20, 20,20,20,1], [100,100,100,100,100,100,1], [500,500,500,500,500,500,1], [20,20,20,20,20,20,20,1] and [100,100,100,100,100,100,100,1]. For a network of size [500,500,500,500,500,500,500,1] we run out of memory on the GPU available to us. This constitutes the limitation of our training. For the optimization we first run the Adam optimizer with learning rate 10^{-4} for 7000 epochs over the initial loss alone. We have found the network struggling to learn the initial condition when the optimization is run on the full loss directly. After the 7000 epochs optimizing the initial loss, we run the Adam optimizer for 50,000 epochs on the full loss function (5.3). Finally, we refine the optimization using L-BFGS.

5.0.3 *Results*

The PDE approximations with FEM and PINNs are compared on the mesh of the ground truth solution spanning [0, 1] with 7993 mesh points and at a time discretization $dt = \frac{1}{3} \times 10^{-4}$ up to time $T = 0.05$. The fine-meshed FEM approximation for the ground truth solution was derived using implicit Euler. The FEM and the PINN approximations compared with the ground truth solution are displayed in Fig. 7 for the different mesh sizes and network architectures. FEM is able to recover the solution of the PDE at all mesh sizes. However, closer inspection of the result for a mesh of size 32 shows slight errors along the diffusive interface. On the other hand, the ability of a PINN to approximate the PDE solution well is dependent on the architecture and the number of free parameters or weights that are to be determined. While all architectures with 20 nodes (cf. Fig. 7 column 1) are not able to recover the solution whatsoever, networks with 100 nodes per layer are able to be trained for the solution. We can clearly observe a progression based on the number of layers. Finally, neural networks that have 500 nodes per layer are all able to approximate the solution well.

Let us compare the solution and evaluation time versus the accuracy for the FEM and PINN approximations as shown in Fig. 8. For the solution time, i.e. the time to solve the PDE using FEM and the time to train the neural network in PINN, FEM is five to six orders of magnitude faster than PINNs. This is mainly due to the size of the neural networks. Here, the complexity of the PDE requires larger architectures to be able to capture the solution. Some of the PINN architectures are then able though to

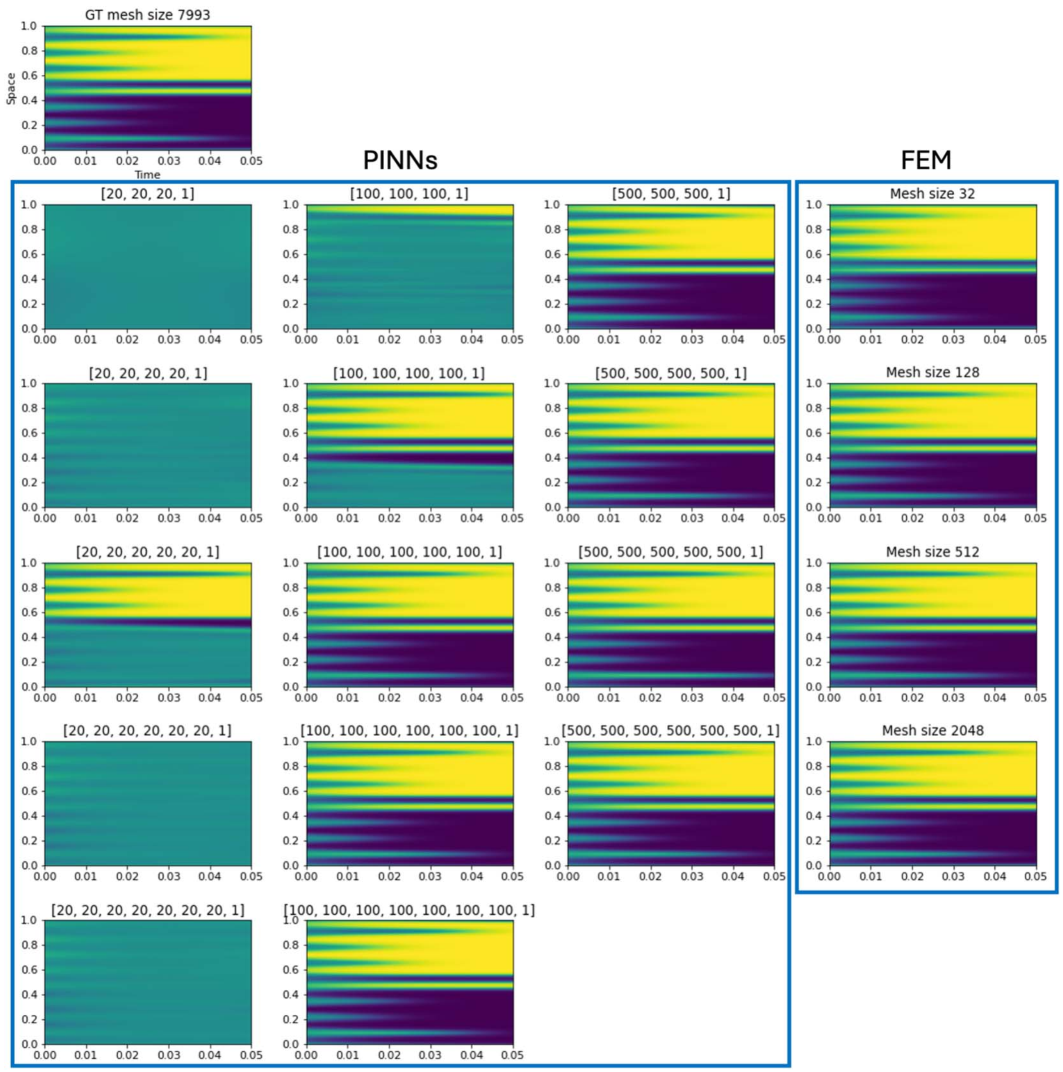
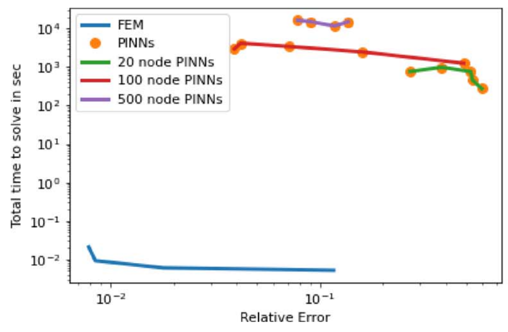


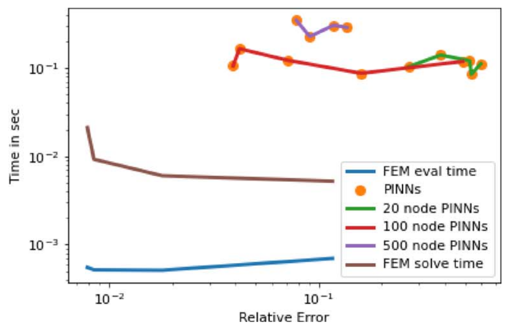
Fig. 7. Comparison of the 1D Allen–Cahn solution approximated by FEM and PINNs.

achieve similar relative errors to FEM. This is especially true for networks with 100 or 500 nodes per layer as also seen in Fig. 7. The evaluation time of the PINN is plotted against the evaluation time of FEM and the solution time of FEM in Fig. 8(b). The calculation time advantage of FEM evaluation drops to about one order of magnitude as compared with the solution time. While it is to be expected that the evaluation of a neural network is much faster then the training, FEM is still faster in both solution and evaluation time.

Compared with the other PDEs that we are considering in this study, the Allen–Cahn equation needs slight modification, i.e. weighting of the loss and pre-training on only part of the loss functional. The need for modification is due to the difficulty we have found the network to have in learning the PDE solution. In addition, we should note that we have also attempted to train a PINN for Allen–Cahn (5.1)



(a) Plot of time to solve FEM and train PINN in sec versus ℓ^2 relative error.



(b) Plot of time to interpolate FEM and evaluate PINN in sec versus ℓ^2 relative error. For comparison, the time to solve FEM is plotted.

FIG. 8. Plot for 1D Allen–Cahn equation of time in sec versus ℓ^2 relative error.

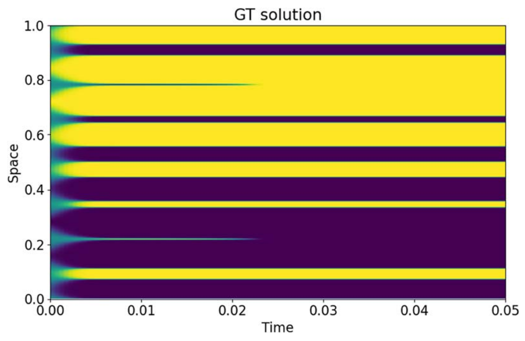


FIG. 9. FEM solution of the 1D Allen–Cahn equation for $\epsilon = 0.001$ derived on a mesh with 7993 cells.

with $\epsilon = 0.001$. The resulting solution—shown in Fig. 9—becomes close to binary after a certain amount of time. We had discussed this effect in Section 2. This renders very large gradients in the solutions. We trained PINNs with different activation functions such as softplus or ReLU that are typically able to handle discontinuities. However, all results were insufficient to be considered an approximation. This speaks to the assumption that PINNs in the vanilla form are not well equipped to handle discontinuous solutions; this may also be due to PINNs solving the strong PDE, rather than a weak form. Variations of the vanilla PINNs approach might be able to obtain satisfactory approximations. However, as we are only considering the vanilla approach, this goes beyond the scope of the paper. We make a note that FEM is able to approximate Allen–Cahn with $\epsilon = 0.001$ albeit we should anticipate the use of a finer mesh to accurate represent the diffuse interface.

6. Approximating the semilinear Schrödinger equation

Finally, we investigate the semilinear Schrödinger equation in one and two space-dimensions. The specifications of the one-dimensional case are taken from the original PINNs paper of Raissi *et al.* (Raissi *et al.*, 2019). Note that the semilinear Schrödinger equation has a complex-valued solution; further increasing the difficulty of its approximation.

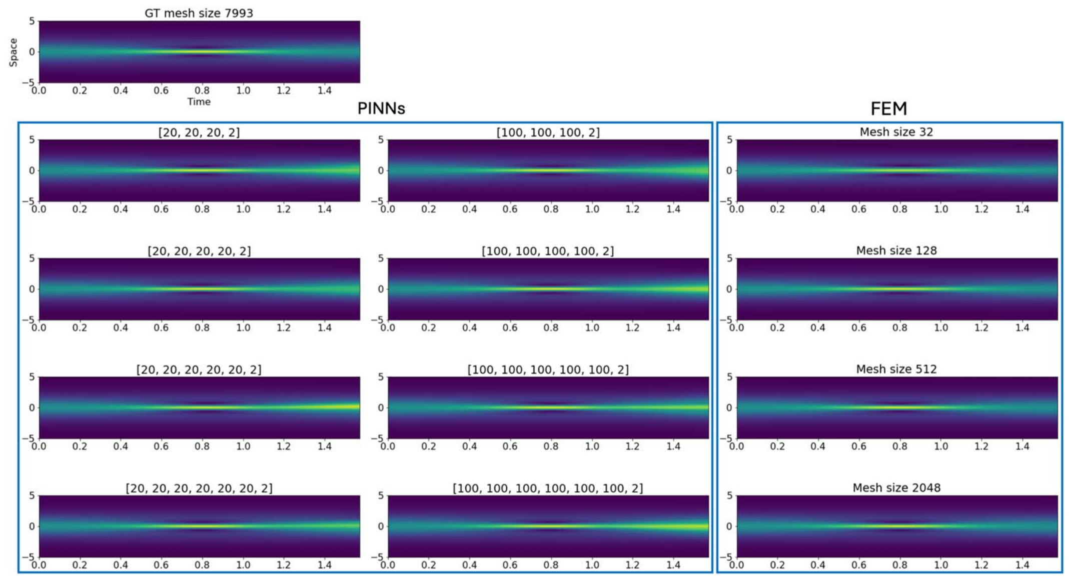


FIG. 10. Comparison of the 1D semilinear Schrödinger solution $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ approximated by FEM and PINNs of different mesh and architecture sizes.

6.1 *1D*

In the one-dimensional case, we consider the semilinear Schrödinger equation with periodic boundary conditions, as follows:

$$
\begin{aligned}
\mathrm{i}\frac{\partial h(t,x)}{\partial t} &= -0.5\Delta h(t,x) - |h(t,x)|^2 h(t,x) && x \in [-5,5],\ t \in [0,\pi/2],\\
h(0,x) &= 2\operatorname{sech}(x), && x \in [-5,5],\\
h(t,-5) &= h(t,5), && t \in [0,\pi/2],\\
\frac{\partial h(t,-5)}{\partial x} &= \frac{\partial h(t,5)}{\partial x}, && t \in [0,\pi/2].
\end{aligned}
$$

We note that the identical problem had also been considered by Raissi *et al.* (2019). As $h(t,x)$ is a complex valued function, the semilinear Schrödinger equation is solved for $h(t,x) =: u_R(t,x) + \mathrm{i}u_I(t,x)$, with $u_R(t,x)$ the real part and $u_I(t,x)$ the imaginary part. Results are visualized for $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ in Fig. 10.

6.1.1 *FEM.* Similar to the 1D Allen–Cahn equation that we considered, we use a semi-implicit Euler strategy to approximate the 1D Schrödinger equation in time. We can then write the weak formulation for the real and imaginary parts of the PDE, assuming Dirichlet $u_\partial = 0$ boundaries, separately as

$$
\begin{aligned}
\int_{-5}^{5}(u_{k+1}^I(x) - u_k^I(x))v^R(x)\mathrm{d}x - 0.5\int_{-5}^{5}\langle\nabla u_{k+1}^R(x), \nabla v^R(x)\rangle\mathrm{d}x - |h_k(x)|^2\int_{-5}^{5}u_{k+1}^R(x)v^R(x)\mathrm{d}x = 0,\\
\int_{-5}^{5}(u_{k+1}^R(x) - u_k^R(x))v^I(x)\mathrm{d}x + 0.5\int_{-5}^{5}\langle\nabla u_{k+1}^I(x), \nabla v^I(x)\rangle\mathrm{d}x + |h_k(x)|^2\int_{-5}^{5}u_{k+1}^I(x)v^I(x)\mathrm{d}x = 0,
\end{aligned}
$$

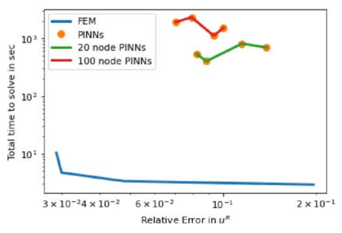
for all test functions $v^R, v^I \in H_0^1(\Omega)$. The time is discretized with $dt = 5 \times 10^{-4}$ and we define the finite elements on an interval mesh in $[-5,5]$. We consider meshes with $n \in \{32, 128, 512, 2048\}$ cells. We again employ piecewise linear finite element basis functions on those cells (P_1) and use the generalized minimal residual method (gmres) to solve the linear problems in the semi-implicit scheme.

6.1.2 *PINNs.* Let us now define the loss functional used in the neural network approximation of the PDE solution. Again, we consider the residual of the PDE, inital condition and boundary conditions as follows:

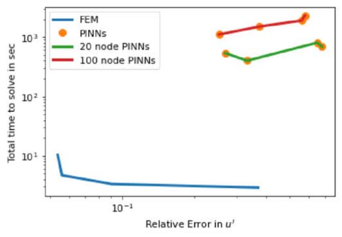
$$\begin{aligned}
\mathrm{Loss}(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}\Big(&\Big\|\frac{\partial u_\theta^I(t_f^i,x_f^i)}{\partial t} - \epsilon \Delta u_\theta^R(t_f^i,x_f^i) - |h_\theta(t_f^i,x_f^i)|^2 u_\theta^R(t_f^i,x_f^i)\Big\|_2^2 \\
&+ \Big\|\frac{\partial u_\theta^R(t_f^i,x_f^i)}{\partial t} + \epsilon \Delta u_\theta^I(t_f^i,x_f^i) + |h_\theta(t_f^i,x_f^i)|^2 u_\theta^I(t_f^i,x_f^i)\Big\|_2^2\Big) \\
+ \frac{1}{N_g}\sum_{k=1}^{N_g}&\left(\|u_\theta^R(t_g^j,-5) - u_\theta^R(t_g^j,5)\|_2^2 + \|u_\theta^I(t_g^j,-5) - u_\theta^I(t_g^j,5)\|_2^2\right) \\
+ \frac{1}{N_g}\sum_{k=1}^{N_g}&\left(\left\|\frac{\partial u_\theta^R(t_g^j,-5)}{\partial x} - \frac{\partial u_\theta^R(t_g^j,5)}{\partial x}\right\|_2^2 + \left\|\frac{\partial u_\theta^I(t_g^j,-5)}{\partial x} - \frac{\partial u_\theta^I(t_g^j,5)}{\partial x}\right\|_2^2\right) \\
+ \frac{1}{N_h}\sum_{k=1}^{N_h}&\left(\|u_\theta^R(0,x_h^k) - 2\operatorname{sech}(x_h^k)\|_2^2 + \|u_\theta^I(0,x_h^k)\|_2^2\right),
\end{aligned}$$

where $h_\theta(t,x) = [u_\theta^R(t,x), u_\theta^I(t,x)]$ is the neural network the produces the real and imaginary parts of the PDE solution of the network output with weights θ. The network is trained on $N_f = 20,000$ collocation points $(t_f^i, x_f^i) \in [0,\pi/2] \times [-5,5]$ and with $N_g = 50$ point on the boundary and $N_h = 50$ points for the initial condition using Latin Hybercube sampling. The network architecture is a feed-forward dense neural network with tanh activation function. In these specifications, we have followed the original vanilla PINNs paper (Raissi *et al.*, 2019). We investigate the performance of eight network architectures: [20,20,20,2], [100,100,100,2], [20,20,20,20,2], [100,100,100,100,2], [20,20,20,20,20,2], [100,100,100,100,100,2], [20,20,20,20,20,20,2] and [100,100,100,100,100,100,2]. We employ the Adam optimizer for 50,000 epochs and a learning rate of 10^{-4}. Afterwards, we use the L-BFGS optimizer to refine the training results.

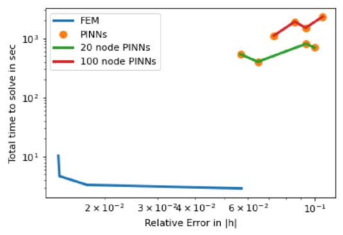
6.1.3 *Results.* The results of the FEM and PINNs approximation are compared on the mesh of the ground truth solution that has a size of 7993 cells and a time discretization with $dt = \frac{1}{3} \times 10^{-4}$. The resulting approximations for $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ are displayed in Fig. 10. Already visually, we notice that the PINN approximations become slightly less accurate for larger time instance than the FEM approximations. Quantitatively speaking, the time versus accuracy plots give a broader overview of the performance of each method. The plots are shown in Fig. 11. Focusing on the modulus $|h|$, FEM both has a lower solution time by two orders of magnitude and a lower relative error than any neural network approximation as shown in Fig. 11(c). Considering the evaluation time for both methods alone, FEM continues to outperform PINNs in time and accuracy. However, the PINN evaluation time is faster than



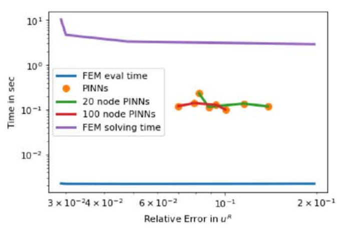
(a) Solution time versus accuracy for the real part of the PDE u^R.



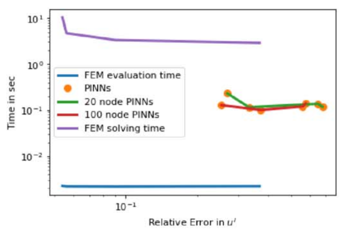
(b) Solution time versus accuracy for the imaginary part of the PDE u^I.



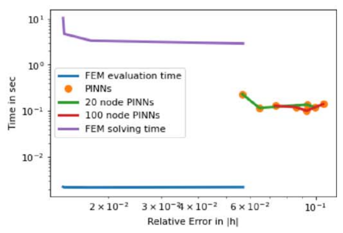
(c) Solution time versus accuracy for the modulus $|h|$.



(d) Evaluation time versus accuracy for the real part of the PDE u^R. For comparison, the time to solve FEM is plotted.



(e) Evaluation time versus accuracy for the imaginary part of the PDE u^I. For comparison, the time to solve FEM is plotted.



(f) Evaluation time versus accuracy for the modulus $|h|$. For comparison, the time to solve FEM is plotted.

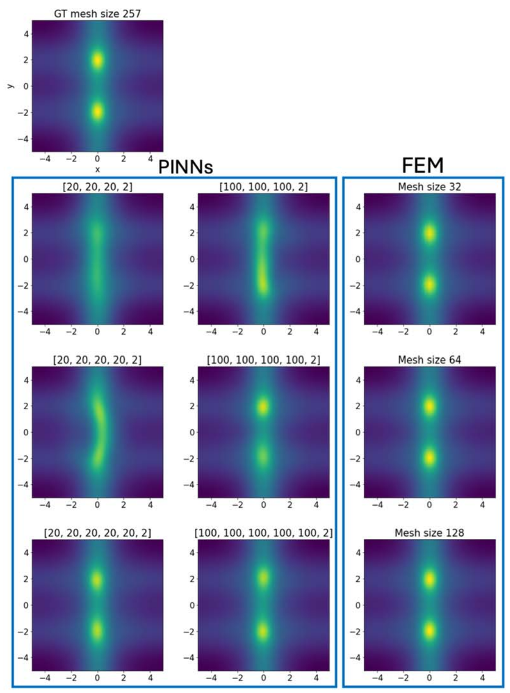
FIG. 11. Plot for 1D Schrödinger equation of time in sec versus ℓ^2 relative error. Plots are split for the real and imaginary parts of the PDE as well as $|h| = \sqrt{u_R^2 + u_I^2}$.

the solution time of FEM. Nonetheless, FEM remains to produce results with higher accuracy for $|h|$. The approximation results are slightly less accurate for u_I. Also there, however, the FEM solution reaches the same or a better accuracy than the PINNs solution at a much faster pace, as we see in Fig. 11(b) and 11(e).

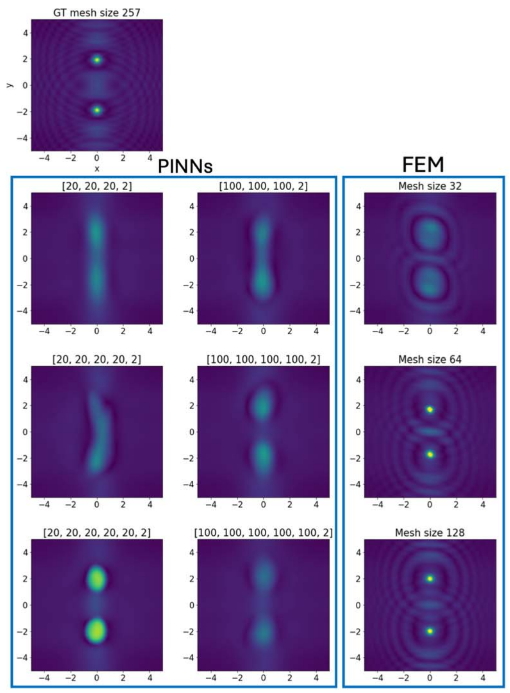
6.2 *2D*

Let us finally move to the two-dimensional semilinear Schrödinger equation. We consider periodic boundary conditions and define the initial condition as follows:

$$\begin{aligned}
\mathrm{i}\frac{\partial h(t,x,y)}{\partial t} &= -0.5\Delta h(t,x,y) - |h(t,x,y)|^2 h(t,x,y), && x,y \in [-5,5],\ t \in [0,\pi/2]\\
h(0,x,y) &= \operatorname{sech}(x) + 0.5\operatorname{sech}(y-2) + 0.5\operatorname{sech}(y+2), && x,y \in [-5,5]\\
h(t,-5,y) &= h(t,5,y), && t \in [0,\pi/2],\ y \in [-5,5]\\
h(t,x,-5) &= h(t,x,5), && t \in [0,\pi/2],\ x \in [-5,5]\\
\frac{\partial h(t,-5,y)}{\partial x} &= \frac{\partial h(t,5,y)}{\partial x}, && t \in [0,\pi/2],\ y \in [-5,5]\\
\frac{\partial h(t,x,-5)}{\partial y} &= \frac{\partial h(t,x,5)}{\partial y} && t \in [0,\pi/2],\ x \in [-5,5].
\end{aligned}$$



(a) $t = \pi/4$



(b) $t = \pi/2$

FIG. 12. Comparison of the 2D semilinear Schrödinger solution $|h(t,x)| = \sqrt{u_R^2(t,x) + u_I^2(t,x)}$ at times $t = \pi/4$ and $t = \pi/2$ approximated by FEM and PINNs of different mesh and architecture sizes.

The PDE is complex-valued and we approximate the solution for $h(t,x,y) =: u_R(t,x,y) + \mathrm{i}u_I(t,x,y)$. The approximate solutions for the modulus for $|h(t,x,y)| = \sqrt{u_R^2(t,x,y) + u_I^2(t,x,y)}$ are shown in Fig. 12.

6.2.1 *FEM.* Similar to the one-dimensional case of the Schrödinger equation, we split the weak formulation of the PDE (again for Dirichlet $u_\partial = 0$ boundaries) into its real and imaginary parts:

$$\int_{-5}^{5}\int_{-5}^{5}(u_{k+1}^I(x,y) - u_k^I(x,y))v^R(x,y)\mathrm{d}x\mathrm{d}y - 0.5\int_{-5}^{5}\int_{-5}^{5}\langle\nabla u_{k+1}^R(x,y), \nabla v^R(x,y)\rangle\mathrm{d}x\mathrm{d}y$$
$$- |h_k(x,y)|^2\int_{-5}^{5}\int_{-5}^{5}u_{k+1}^R(x,y)v^R(x,y)\mathrm{d}x\mathrm{d}y = 0,$$
$$\int_{-5}^{5}\int_{-5}^{5}(u_{k+1}^R(x,y) - u_k^R(x,y))v^I(x,y)\mathrm{d}x\mathrm{d}y + 0.5\int_{-5}^{5}\int_{-5}^{5}\langle\nabla u_{k+1}^I(x,y), \nabla v^I(x,y)\rangle\mathrm{d}x\mathrm{d}y$$
$$+ |h_k(x,y)|^2\int_{-5}^{5}\int_{-5}^{5}u_{k+1}^I(x,y)v^I(x)\mathrm{d}x\mathrm{d}y = 0,$$

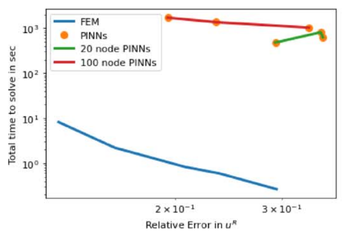
for all test functions $v^R, v^I \in H_0^{\Omega}$. Time is discretized with $dt = 10^{-3}$ and the finite elements are defined on a rectangular mesh in the domain $[-5,5] \times [-5,5]$. We consider meshes with $\{(16,16),(32,32),(40,40),(64,64),(128,128)\}$ squares, each again being split into two triangles on which we define piecewise-linear finite element basis functions (P_1). We again use gmres to solve the linear system.

6.2.2 *PINNs.* The loss functional for the neural network approximations are defined for the real and imaginary parts:

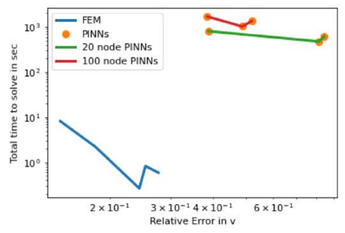
$$\begin{aligned}
\text{Loss}(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}(\| &\frac{\partial u_\theta^I(t_f^i,x_f^i,y_f^i)}{\partial t} - \epsilon \Delta u_\theta^R(t_f^i,x_f^i,y_f^i) - |h_\theta(t_f^i,x_f^i,y_f^i)|^2 u_\theta^R(t_f^i,x_f^i,y_f^i)\|_2^2 \\
&+ \|\frac{\partial u_\theta^R(t_f^i,x_f^i,y_f^i)}{\partial t} + \epsilon \Delta u_\theta^I(t_f^i,x_f^i,y_f^i) + |h_\theta(t_f^i,x_f^i,y_f^i)|^2 u_\theta^I(t_f^i,x_f^i,y_f^i)\|_2^2) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}(\|u_\theta^R(t_g^j,-5,y_g^j) - u_\theta^R(t_g^j,5,y_g^j)\|_2^2 + \|u_\theta^I(t_g^j,-5,y_g^j) - u_\theta^I(t_g^j,5,y_g^j)\|_2^2 \\
&\quad + \|u_\theta^R(t_g^j,x_g^j,-5) - u_\theta^R(t_g^j,x_g^j,5)\|_2^2 + \|u_\theta^I(t_g^j,x_g^j,-5) - u_\theta^I(t_g^j,x_g^j,5)\|_2^2) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}\left(\left\|\frac{\partial u_\theta^R(t_g^j,-5,y_g^j)}{\partial x} - \frac{\partial u_\theta^R(t_g^j,5,y_g^j)}{\partial x}\right\|_2^2 + \left\|\frac{\partial u_\theta^I(t_g^j,-5,y_g^j)}{\partial x} - \frac{\partial u_\theta^I(t_g^j,5,y_g^j)}{\partial x}\right\|_2^2\right) \\
&+ \frac{1}{N_g}\sum_{k=1}^{N_g}\left(\left\|\frac{\partial u_\theta^R(t_g^j,x_g^j,-5)}{\partial y} - \frac{\partial u_\theta^R(t_g^j,x_g^j,5)}{\partial y}\right\|_2^2 + \left\|\frac{\partial u_\theta^I(t_g^j,x_g^j,-5)}{\partial y} - \frac{\partial u_\theta^I(t_g^j,x_g^j,5)}{\partial y}\right\|_2^2\right) \\
&+ \frac{1}{N_h}\sum_{k=1}^{N_h}\left(\|u_\theta^R(0,x_h^k,y_h^k) - \text{sech}(x_h^k) - 0.5\text{sech}(y_h^k-2) - 0.5\text{sech}(y_h^k+2)\|_2^2\right),
\end{aligned}$$

for $h_\theta(t,x,y) = u_\theta^R(t,x,y) + \mathrm{i}u_\theta^I(t,x,y)$ the neural network with an output size 2 for the real and imaginary parts and θ the network weights that are determined by training. The loss functional is optimized based on latin hybercube sampled collocation points $(t_f^i,x_f^i,y_f^i) \in [0,\pi/2]\times[-5,5]\times[-5,5]$ with $N_f = 5{,}000$ points for the domain, $N_g = 100$ points for the boundary and $N_h = 100$ points for the initial condition. We chose feed-forward dense neural network in the architecture design with $\tanh$ the activation function. The results are compared for six different neural network sizes: [20,20,20,2], [100,100,100,2], [20,20,20,20,2], [100,100,100,100,2], [20,20,20,20,20,2] and [100,100,100,100,100,2]. The Adam optimizer is run for 50,000 epochs with a learning rate of 10^{-3} before employing the L-BFGS optimizer.

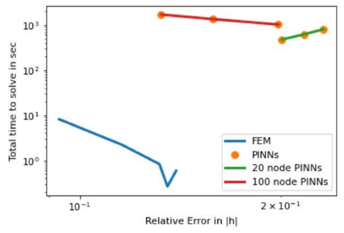
6.2.3 *Results.* The ground truth solution was derived with FEM on a mesh of 257 cells using P_2 basis functions. The time stepping size up to $T = \pi/2$ was chosen as $dt = 5 \times 10^{-4}$. A first look at the visualization of the results at an example times $t = \pi/4$ and $\pi/2$ in Fig. 12 shows that PINNs is having difficulties to recover particularly the wave-type shapes and the correct diameter of the peaks at $t = \pi/2$. Smaller network architectures are not always able to accurately separate the two peaks in the image even



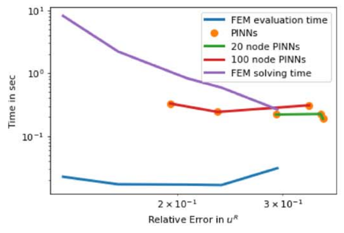
(a) Solution time versus accuracy for the real part of the PDE u^R.



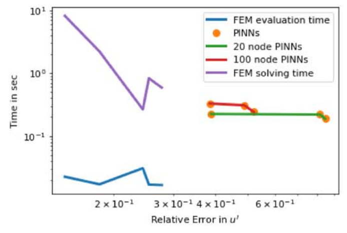
(b) Solution time versus accuracy for the imaginary part of the PDE u^I.



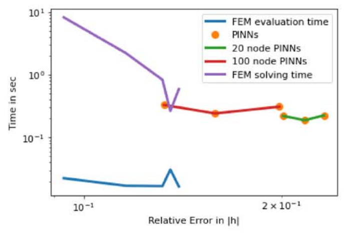
(c) Solution time versus accuracy for the modulus $|h|$.



(d) Evaluation time versus accuracy for the real part of the PDE u^R. For comparison, the time to solve FEM is plotted.



(e) Evaluation time versus accuracy for the imaginary part of the PDE u^I. For comparison, the time to solve FEM is plotted.



(f) Evaluation time versus accuracy for modulus $|h|$. For comparison, the time to solve FEM is plotted.

FIG. 13. Plot for 2D Schrödinger equation of time in sec versus ℓ^2 relative error. Plots are split for the real and imaginary parts of the PDE as well as $|h| = \sqrt{u_R^2 + u_I^2}$.

at earlier time instances. While PINNs are able to reproduce the main features of the PDE solution when the architecture is large enough, the examples shown lack the detailed features containing edges and discontinuities. For FEM, we observe that coarse meshes similarly fail at recovering the fine details. We also observe that the symmetry of the peaks is not preserved with a coarser mesh. However, with finer meshes, the FEM approximation appears to be good. Considering the quantified results on time versus accuracy in Fig. 13, the difference in accuracy between PINNs and FEM becomes less prudent in the real part. However, for both solving time in Figs 11(a)–13(c) and evaluation time in Figs 11(d)–13(f) FEM clearly outperforms PINNs by two to three and one order of magnitude, respectively. Taking the FEM solution time into account, we can see that PINNs slightly outperforms in their evaluation time with similar relative error. Let us also note that the PINN approximation of the imaginary part of the complex-valued PDE solution is significantly less accurate compared with the FE method.

7. Discussion and conclusions

After having investigated each of the PDEs on its own, let us now discuss and draw conclusions from the results as a whole. Considering the solution time and accuracy, PINNs are not able to beat FEM in our study. In all the examples that we have studied, the FEM solutions were faster at the same or at a higher accuracy.

We will now try to explain this outcome by briefly discussing the computational complexity of the FEM and of the training in PINNs. When using a multigrid method, an elliptic equation can often be solved in $O(N_{\text{DoF}})$ floating point operations, where N_{DoF} is the number of degrees of freedom in the discretization, see, e.g. chapter 13 in Iserles (2008). We usually expect N_{DoF} to increase exponentially with the dimension d. We have neither used geometric nor algebraic multigrid methods in this work; the algorithms we used are actually slower than multigrid, but we still argue with the theoretical cost of multigrid methods. If we ignore the GPU architecture, we can estimate the complexity of a single gradient evaluation with respect to weights and biases and give one data point in the PINN as

$$O(dN_e + N_l N_e^2 + N_e), \tag{7.1}$$

where $N_l \in \mathbb{N}$ describes the number of hidden layers, i.e. all layers but the input and the output layer. To simplify the presentation, we make the assumption that $N_e(l) = N_e$ constant for all layers $l = 1, ..., N_l$ and $N_e \in \mathbb{N}$. The first layer is of size $d \times N_e$, the last one is of size $N_e \times 1$ and all intermediate hidden layers are of size $N_e \times N_e$. We have derived this complexity following the neat summary of the backpropagation algorithm in Higham & Higham (2019), where we note that gradient evaluations in PINNs actually require multiple backpropagations for each step, but this finite number does not increase the asymptotic complexity. When employing the Adam optimizer, the complexity cost of a single step without mini-batching is exactly the given cost in (7.1); the cost for one Adam epoch is given by $N_f + N_g + N_h$ times the cost in (7.1). In L-BFGS, we similarly incur the cost of one Adam epoch in each step and additionally linear cost in the size of the parameter space that does not change the asymptotics. Thus, denoting the number of Adam epochs and L-BFGS steps by N_{Adam} and $N_{\text{L-BFGS}}$, respectively, the complete cost can be stated as

$$O((dN_e + N_l N_e^2 + N_e)(N_{\text{Adam}} + N_{\text{L-BFGS}})(N_f + N_g + N_h)).$$

This appears to scale nicely in dimension d, but naturally $N_l, N_e, N_{\text{Adam}}, N_{\text{L-BFGS}}, N_f, N_g, N_h$ may depend on d. The usual GPU acceleration of backpropagation reduces the computational cost of these operations considerably. Moreover, especially in high dimensions, we would usually anticipate to be able to choose number and size of layers such that the combined size of weight matrices $dN_e + N_l N_e^2 + N_e$ and bias vectors $N_l N_e + 1$ is considerably smaller than the degrees of freedom N_{DoF} in a usual FEM approximation. The largest contribution to the computational cost is likely the large number of necessary Adam and L-BFGS steps that are necessary in the non-convex learning problem.

After having solved the PDE, PINNs are sometimes faster at the pointwise evaluation of the respective solution: we were only able to show this in the 3D Poisson test, where the FEM evaluation after solution was rather slow. So when needing to evaluate a PDE on a very fine grid, one could consider solving a PINN. Although, in our examples, the solution time of FEM was so much faster that continued solving the PDE with FEM on different adapted grids would likely still be considerably faster than solving and evaluating the PINN. In addition, we believe that the evaluation time in FEM could be significantly sped up using more appropriate interpolation methodology, such as Lin & Lee (2005); Zhang *et al.* (2021).

We were particularly surprised that PINNs had difficulties with the Allen–Cahn equation using a small ε. This might be due to the close-to-discontinuous behaviour at the diffuse interface, which may be more pronounced in the strong form of the PDE that PINNs aim to solve. We anticipated that PINNs would outperform FEM: the solution of an Allen–Cahn equation has very much the flavour of a classification (see Budd *et al.* (2021)) at which neural networks excel; FEM requires a very fine grid to resolve the diffuse interface. A similar case in which we were surprised that PINNs did not outperform FEM was the Schrödinger 2D examples, where the PDE solution, again, contains very finely structured areas. In both these cases an adaptive PINNs approach or variational PINNs (Kharazmi *et al.*, 2019) might help. The latter would then also allow activation functions that have weak derivatives.

An aspect of the evaluation time that we have not considered throughout this work is the possibility of solving parameterized PDEs with neural networks, the so-called operator approximators. See, for instance, Fourier Neural Operators (Li *et al.*, 2021) and DeepONets (Lu *et al.*, 2021a). While FEM requires continued solutions of PDEs when changing the parameters, neural networks can take parameters as additional inputs and be trained throughout all of them—we have seen that PINN evaluations are sometimes faster than FEM solutions. They have been shown to work well as surrogates if the PDE needs to be solved sufficiently often; see also Tanyu *et al.* (2023). In a future work, those should be compared with classical methods for parameterized PDEs, such as reduced bases (Quarteroni *et al.*, 2016) or low-rank tensor methods (Kressner & Tobler, 2011). Again, the time of the offline phase in which the parametric model is constructed or trained has to be considered carefully.

PINNs were good at the transition into higher dimensions: there is no increment in computational cost from the Poisson equation in 2D and 3D. The 3D Poisson equation we consider is completely isotropic, which is a structure the PINN appears to be able to use easily. A basic FEM approach cannot make use of it. In a more complicated anisotropic or even regular spatially varying setting, we would anticipate a similar computational cost and speed for FEM, but cannot make any predictions for PINNs. This hints at the efficiency of PINNs in certain high-dimensional settings, in which classical techniques (such as FEM) are prohibitively expensive. This has been considered in, e.g. Kunisch & Walter (2021) and Hu *et al.* (2024). In general, PINNs open up many interesting new research directions, especially when employed to solve such high-dimensional PDEs or when combining PDEs and data. The analysis of PINNs is both very challenging and highly interesting. Our study suggests that for certain classes of PDEs for which classical methods are applicable, PINNs are not able to outperform those.

Acknowledgements

Initial research for this work was carried out with support from the Philippa Fawcett internship programme. The authors thank the anonymous reviewers for several comments that improved this manuscript. The authors are also grateful for remarks by Bernhard Heinzelreiter, Rainald Lohner, John Pearson, David Silvester and Marcus Webb. We also acknowledge the support of NVIDIA Corporation with the donation of two Quadro P6000 GPU used for this research.

Funding

Cantab Capital Institute for the Mathematics of Information (T.G.G. and C.B.S.); European Union Horizon 2020 research and innovation programme (777826 NoMADS to T.G.G. and C.B.S.); EPSRC National Productivity and Investment Fund (EP/S515334/1 reference 2089694 to T.G.G.); EPSRC (EP/S026045/1 to J.L. and C.B.S.); Philip Leverhulme Prize (C.B.S.); the Royal Society Wolfson Fellowship (C.B.S.); the EPSRC advanced career fellowship (EP/V029428/1 to C.B.S.); EPSRC (EP/T003553/1, EP/N014588/1, EP/T017961/1 to C.B.S.); the Wellcome Trust (215733/Z/19/Z and 221633/Z/20/Z to C.B.S.); the Alan Turing Institute (C.B.S.).

References

Allen, S. M. & Cahn, J. W. (1972) Ground state structures in ordered binary alloys with second neighbor interactions. *Acta Metall.*, **20**, 423–433.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E. & Wells, G. N. (2015) The FEniCS Project Version 1.5. *Arch. Numer. Softw.*, **3**, 9–23.

Baydin, A. G., Pearlmutter, B. A., Radul, A. A. & Siskind, J. M. (2018) Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, **18**, 1–43.

Beirão da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L. D. & Russo, A. (2013) Basic principles of virtual element methods. *Math. Models Methods Appl. Sci.*, **23**, 199–214.

Bellmann, R. (1954) Dynamic programming and a new formalism in the calculus of variations. *Proc. Natl. Acad. Sci.*, **40**, 231–235.

Beneš, M., Chalupecký, V. & Mikula, K. (2004) Geometrical image segmentation by the Allen–Cahn equation. *Appl. Numer. Math.*, **51**, 187–205.

Bertozzi, A. & Schönlieb, C.-B. (2010) Unconditionally stable schemes for higher order inpainting. *Commun. Math. Sci.*, **9**, 413–457.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S. & Zhang, Q. (2018) JAX: composable transformations of Python+NumPy programs.

Braess, D. (2007) *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, 3rd edn. Cambridge: Cambridge University Press.

Budd, J., van Gennip, Y. & Latz, J. (2021) Classification and image processing with a semi-discrete scheme for fidelity forced Allen–Cahn on graphs. *GAMM-Mitteilungen*, **44**, e202100004.

Bungartz, H.-J. & Griebel, M. (2004) Sparse grids. *Acta Numer.*, **13**, 147–269.

Burger, M., Caffarelli, L. & Markowich, P. A. (2014) Partial differential equation models in the socio-economic sciences. *Philos Trans A Math Phys. Eng Sci*, **372**, 20130406.

Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D. & Di Giovanni, M. (2020) NeuroDiffEq: A Python package for solving differential equations with neural networks. *J. Open Source Softw.*, **5**, 1931.

Chuang, P.-Y. & Barba, L. A. (2022) Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration. arXiv preprint arXiv:2205.14249.

Courant, R. (1943) Variational methods for the solution of problems of equilibrium and vibrations. *Bull. Am. Math. Soc.*, **49**, 1–23.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M. & Piccialli, F. (2022) Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next. *J. Sci. Comput.*, **92**, 88.

De Ryck, T., Jagtap, A. D. & Mishra, S. (2024) Error estimates for physics-informed neural networks approximating the navier–stokes equations. *IMA J. Numer. Anal.*, **44**, 83–119.

Egger, H., Rüde, U. & Wohlmuth, B. (2014) Energy-corrected finite element methods for corner singularities. *SIAM J. Numer. Anal.*, **52**, 171–193.

Eymard, R., Gallouët, T. & Herbin, R. (1997) Finite Volume Methods. *Handbook of Numerical Analysis* (P. G. Ciarlet & J. L. Lions eds), vol. **7**. Elsevier, pp. 713–1020.

Fabiani, G., Calabrò, F., Russo, L. & Siettos, C. (2021) Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines. *J. Sci. Comput.*, **89**, 44.

Feng, X. & Prohl, A. (2003) Numerical analysis of the Allen–Cahn equation and approximation for mean curvature flows. *Numer. Math.*, **94**, 33–65.

Feng, X. & Hai-jun, W. (2005) A Posteriori Error Estimates and an Adaptive Finite Element Method for the Allen–Cahn Equation and the Mean Curvature Flow. *J. Sci. Comput.*, **24**, 121–146.

Hadamard, J. (1902) Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 49–52.

Hennigh, O., Narasimhan, S., Nabian, M. A., Subramaniam, A., Tangsali, K., Fang, Z., Rietmann, M., Byeon, W. & Choudhry, S. (2021) Nvidia simnet$^{\text{TM}}$: An ai-accelerated multi-physics simulation framework. *Computational Science – ICCS 2021* (M. Paszynski, D. Kranzlmüller, V. V. Krzhizhanovskaya, J. J. Dongarra & P. M. A. Sloot eds). Cham: Springer International Publishing, pp. 447–461.

Heston, S. L. (2015) A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Rev. Financ. Stud.*, **6**, 327–343.

Higham, C. F. & Higham, D. J. (2019) Deep learning: An introduction for applied mathematicians. *SIAM Rev.*, **61**, 860–891.

Hrennikoff, A. (2021) Solution of problems of elasticity by the framework method. *J. Appl. Mech.*, **8**, A169–A175.

Hu, Z., Shukla, K., Karniadakis, G. E. & Kawaguchi, K. (2024) Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 106369.

Hulbert, G. M. & Hughes, T. J. R. (1990) Space-time finite element methods for second-order hyperbolic equations. *Comput. Methods Appl. Mech. Eng.*, **84**, 327–348.

Iserles, A. (2008) *A First Course in the Numerical Analysis of Differential Equations. Cambridge Texts in Applied Mathematics*, 2nd edn. Cambridge: Cambridge University Press.

Jagtap, A. D. & Karniadakis, G. E. (2020) Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Commun. Comput. Phys.*, **28**, 2002–2041.

Jagtap, A. D., Kharazmi, E. & Karniadakis, G. E. (2020) Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.*, **365**, 113028.

Jin, K., Latz, J., Liu, C. & Schönlieb, C.-B. (2023) A continuous-time stochastic gradient descent method for continuous data. *J. Mach. Learn. Res.*, **24**, 1–48.

Kharazmi, E., Zhang, Z. & Karniadakis, G. E. (2019) Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. arXiv preprint arXiv:1912.00873.

Kharazmi, E., Zhang, Z. & George, E. M. (2021) Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.*, **374**, 113547.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, 2015.

Kiran, U., Sharma, D. & Gautam, S. S. (2023) A gpu-based framework for finite element analysis of elastoplastic problems. *Computing*, **105**, 1673–1696.

Koto, T. (2008) Imex runge–kutta schemes for reaction–diffusion equations. *J. Comput. Appl. Math.*, **215**, 182–195.

Kovacs, A., Exl, L., Kornell, A., Fischbacher, J., Hovorka, M., Gusenbauer, M., Breth, L., Oezelt, H., Yano, M., Sakuma, N., Kinoshita, A., Shoji, T., Kato, A. & Schrefl, T. (2022) Conditional physics informed neural networks. *Commun. Nonlinear Sci. Numer. Simul.*, **104**, 106041.

Kressner, D. & Tobler, C. (2011) Low-rank tensor Krylov subspace methods for parametrized linear systems. *SIAM J. Matrix Anal. Appl.*, **32**, 1288–1316.

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R. & Mahoney, M. W. (2021) Characterizing possible failure modes in physics-informed neural networks. *Adv. Neural Inf. Process. Syst.*, **34**, 26548–26560.

Kunisch, K. & Walter, D. (2021) *Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation*, vol. **27**. ESAIM: COCV, p. 16.

Kushner, H. J. (1964) On the differential equations satisfied by conditional probablitity densities of markov processes, with applications. *J.o Soc. Ind. Appl. Math. A Control*, **2**, 106–119.

Lawrence, C. (2010) *Evans. Partial Differential Equations*. Providence, R.I: American Mathematical Society.

Li, W., Bazant, M. Z. & Zhu, J. (2021) A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches. *Comput. Methods Appl. Mech. Eng.*, **383**, 113933.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. & Anandkumar, A. (2021) Fourier neural operator for parametric partial differential equations. *9th International Conference on Learning Representations*, ICLR 2021, May 3–7, 2021, Conference Track Proceedings.

Lin, C. C. & Segel, L. A. (1988) Mathematics Applied to Deterministic Problems in the Natural Sciences. *Soc. Ind. Appl. Math.*.

Lin, P.-H. & Lee, T.-Y. (2005) A fast 2D shape interpolation technique. *Computational Science and Its Applications – ICCSA 2005* (O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar & C. J. K. Tan eds). Springer, pp. 1050–1059.

Liu, D. C. & Nocedal, J. (1989) On the limited memory BFGS method for large scale optimization. *Math. Program.*, **45**, 503–528.

Liu, G.-R. (2009) *Meshfree methods: moving beyond the finite element method*. Boca Raton: CRC press.

LOGG, A., MARDAL, K.-A. & WELLS, G. (2012) *Automated Solution of Differential Equations by the Finite Element Method*. Berlin Heidelberg: Springer.

LU, L., JIN, P., PANG, G., ZHANG, Z. & KARNIADAKIS, G. E. (2021a) Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature. Mach. Intell.*, **3**, 218–229.

LU, L., MENG, X., MAO, Z. & KARNIADAKIS, G. E. (2021b) DeepXDE: A deep learning library for solving differential equations. *SIAM Rev.*, **63**, 208–228.

MA, C., LEI, W., et al. (2022) The barron space and the flow-induced function spaces for neural network models. *Constr. Approx.*, **55**, 369–406.

MAO, Z., JAGTAP, A. D. & KARNIADAKIS, G. E. (2020) Physics-informed neural networks for high-speed flows. *Comput. Methods Appl. Mech. Eng.*, **360**, 112789.

MISHRA, S. & MOLINARO, R. (2022) Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. *IMA J. Numer. Anal.*, **42**, 981–1022.

MOSELEY, B., MARKHAM, A. & NISSEN-MEYER, T. (2023) Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Adv. Comput. Math.*, **49**, 62.

PATERA, A. T. (1984) A spectral element method for fluid dynamics: Laminar flow in a channel expansion. *J. Comput. Phys.*, **54**, 468–488.

QUARTERONI, A., MANZONI, A. & NEGRI, F. (2016) *Reduced Basis Methods for Partial Differential Equations: An Introduction*. Cham: Springer International Publishing.

RAHAMAN, N., BARATIN, A., ARPIT, D., DRAXLER, F., LIN, M., HAMPRECHT, F., BENGIO, Y. & COURVILLE, A.. On the Spectral Bias of Neural Networks. In CHAUDHURI, K. and SALAKHUTDINOV, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume **97** of Proceedings of Machine Learning Research, pages 5301–5310. PMLR, 09–15 Jun2019.

RAISSI, M., PERDIKARIS, P. & KARNIADAKIS, G. E. (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, **378**, 686–707.

RAISSI, M., PERDIKARIS, P. & KARNIADAKIS, G. E. (2017a) Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. arXiv preprint arXiv:1711.10561.

RAISSI, M., PERDIKARIS, P. & KARNIADAKIS, G. E. (2017b) *Physics Informed Deep Learning* (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. arXiv preprint arXiv:1711.10566.

RISKEN, H. (1996) *The Fokker-Planck Equation: Methods of Solution and Applications*. Berlin, Heidelberg: Springer.

ROZENMAN, G. G., SHEMER, L. & ARIE, A. (2020) Observation of accelerating solitary wavepackets. *Phys. Rev. E (3)*, **101**, 050201.

RUDIN, L. I., OSHER, S. & FATEMI, E. (1992) Nonlinear total variation based noise removal algorithms. *Physica D*, **60**, 259–268.

SANDER, O. (2020) *DUNE — The Distributed and Unified Numerics Environment*. Cham: Springer International Publishing.

SCHIESSER, W. E. & GRIFFITHS, G. W. (2009) *A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab*. Cambridge: Cambridge University Press.

SHI, D., LIAO, X. & WANG, L. (2016) Superconvergence analysis of conforming finite element method for nonlinear Schrödinger equation. *Appl. Math. Comput.*, **289**, 298–310.

SHIN, Y., DARBON, J. & KARNIADAKIS, G. E. (2020) On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. arXiv preprint arXiv:2004.01806.

SIRIGNANO, J. & SPILIOPOULOS, K. (2018) DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, **375**, 1339–1364.

SMITH, J. D., ROSS, Z. E., AZIZZADENESHELI, K. & MUIR, J. B. (2021) HypoSVI: Hypocentre inversion with Stein variational inference and physics informed neural networks. *Geophys. J. Int.*, **228**, 698–710.

STEIN, M. (1987) Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, **29**, 143–151.

Strauss, W. A. (1978) The Nonlinear Schrödinger Equation. In Guilherme M.De La Penha and Luiz Adauto J. Medeiros, editors, *Contemporary Developments in Continuum Mechanics and Partial Differential Equations*, volume **30** of North-Holland Mathematics Studies, North-Holland: Elsevier, pages 452–465.

Tanyu, D. N., Ning, J., Freudenberg, T., Heilenkötter, N., Rademacher, A., Iben, U. & Maass, P. (2023) Deep learning methods for partial differential equations and related parameter identification problems. *Inverse Probl.*, **39**, 103001.

Taubes, C. H. (2008) *Modeling Differential Equations in Biology*, 2nd edn. Cambridge: Cambridge University Press.

Weinan, E. & Bing, Y. (2018) The Deep Ritz Method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, **6**, 1–12.

Wojtowytsch, S. & Weinan, E. (2020) Can shallow neural networks beat the curse of dimensionality? A mean field training perspective. *IEEE Transactions on. Artif. Intell.*, **1**, 121–129.

Xavier Sierra-Canto, X., Madera-Ramirez, F., & Uc-Cetina, V. (2010) Parallel training of a back-propagation neural network using cuda. In *Ninth International Conference on Machine Learning and Applications*, pages 307–312.

Yang, L., Meng, X. & Karniadakis, G. E. (2021) B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.*, **425**, 109913.

Zhang, N., Canini, K., Silva, S. & Gupta, M. (2021) Fast linear interpolation. *ACM J. Emerging Technol. Comput. Syst.*, **17**, 1–15.